(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開2000－215061
（P2000－215061A）
(43)公開日　平成12年８月４日(2000.8.4)

(51)Int.Cl.'		識別記号	FI		テーマコード'（参考）
Ｇ０６Ｆ	9/38	３７０	Ｇ０６Ｆ　9/38	３７０Ｘ	
	9/30	３１０	9/30	３１０Ａ	
		３５０		３５０Ｆ	

審査請求　未請求　請求項の数32　ＯＬ　外国語出願　（全 64 頁）

(21)出願番号　特願平11－321534

(22)出願日　平成11年10月６日(1999.10.6)

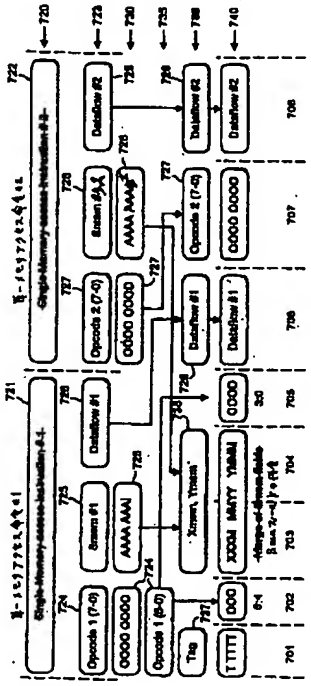
(31)優先権主張番号　９８４０２４５６．２
(32)優先日　平成10年10月６日(1998.10.6)
(33)優先権主張国　欧州特許庁（ＥＰ）

(71)出願人　590000879
テキサス　インスツルメンツ　インコーポレイテッド
アメリカ合衆国テキサス州ダラス，ノース　セントラルエクスプレスウエイ　13500
(72)発明者　ジルベール　ローランティ
フランス国，サン　ポール　ド　バンス，シュマン　ド　サン　エチアンヌ，1490
(74)代理人　100066692
弁理士　浅村　皓　（外３名）

最終頁に続く

(54)【発明の名称】　複合メモリアクセス命令

(57)【要約】
【課題】　処理エンジンの性能を改善する。
【解決手段】　処理エンジンは、実行未決単一複合命令をバッファするように動作する命令バッファを含み、そこからの命令を復号する復号機構が、別々のプログラムされたメモリ命令から形成される複合命令を表す命令のタグフィールド内の所定のタグ７２６に応答して少なくとも第１のプログラムされた命令７２１に対する第１のデータフロー制御および第２のプログラムされた命令７２２に対する第２のデータフロー制御７２９を復号する。複合命令を使用して処理エンジンで利用できる帯域幅を有効利用できる。別々の第１および第２のプログラムされたメモリ命令からソフトデュアルメモリ命令をコンパイルできる。所定の複合命令の複合アドレスフィールド７３８をハード複合メモリ命令すなわちプログラムされる複合命令に対するアドレスフィールドと同じビット位置に配列できる。



1

【特許請求の範囲】

【請求項１】　処理エンジンを含むデジタルシステムであって、

前記処理エンジンが、

その実行が未決である単一の複合命令をバッファするように動作する命令バッファと、

該命令バッファからの命令を復号するように構成されている復号機構と、を含み、

該復号機構が、別々のプログラムされたメモリ命令から形成された複合命令である命令を表す、命令内の所定のタグに応答して、少なくとも第１のプログラムされた命令に対する第１のデータフロー制御および少なくとも第２のプログラムされた命令に対する第２のデータフロー制御を復号する、

デジタルシステム。

【請求項２】　前記複合命令が、別々の第１および第２のプログラムされたメモリ命令を結合することによって形成される複合メモリ命令である、請求項１記載の処理エンジン。

【請求項３】　前記復号機構が、前記複合命令内の複合メモリアドレスフィールドからの第１のプログラムされたメモリアドレス命令に対する第１のメモリアドレスおよび第２のプログラムされたメモリ命令に対する第２のメモリアドレスを復号するように動作する、請求項２記載の処理エンジン。

【請求項４】　前記複合命令の前記複合アドレスフィールドが、ハードプログラムされたデュアルメモリ命令に対する前記アドレスフィールドと同じビット位置である、請求項３記載の処理エンジン。

【請求項５】　前記複合命令の前記複合アドレスフィールド内の前記メモリアドレスが、間接アドレスであり、前記復号機構が、前記間接アドレスを復号するように動作する、

請求項４記載の処理エンジン。

【請求項６】　前記複合命令が、該複合命令の第１のプログラムされた命令に対する分割操作符号フィールドを含む、請求項１記載の処理エンジン。

【請求項７】　前記復号機構が、前記所定のタグに応答して、前記複合命令の前記第１のプログラムされた命令に対する分割操作符号を復号する、請求項６記載の処理エンジン。

【請求項８】　前記複合命令が、その操作符号フィールドが前記第１のプログラムされた命令の前記操作符号フィールドよりも少ないビットを含む前記複合命令の第１のプログラムされた命令に対する操作符号フィールドを含む、請求項７記載の処理エンジン。

【請求項９】　前記復号機構が、前記所定のタグに応答して、前記複合命令の前記第１のプログラムされた命令に対する縮小サイズ操作符号を復号する、請求項８記載の処理エンジン。

2

【請求項１０】　前記複合命令が、前記別々のプログラムされた命令の前記ビットの合計と同じ総ビット数を有する、請求項９記載の処理エンジン。

【請求項１１】　前記複合命令が、前記第１および第２のプログラムされたメモリ命令のデータアドレス発生（ＤＡＧＥＮ）フィールドから形成された結合ＤＡＧＥＮフィールドを有する、請求項１記載の処理エンジン。

【請求項１２】　前記結合ＤＡＧＥＮフィールドが、結合アドレスフィールドの一部を形成する、請求項１１記載の処理エンジン。

【請求項１３】　前記復号機構が、所定のＤＡＧＥＮタグに応答して、前記結合ＤＡＧＥＮフィールドを復号する、請求項１２記載の処理エンジン。

【請求項１４】　前記第１および第２のメモリアドレスによって識別されるアドレスから第１および第２のオペランドを並列にフェッチするように動作するフェッチコントローラを含む、請求項１記載の処理エンジン。

【請求項１５】　前記第１および第２のプログラムされた命令に対する第１および第２のデータフロー操作の結果を並列に書き込むように動作するライトコントローラを含む、請求項１４記載の処理エンジン。

【請求項１６】　メモリアクセス命令を明確に並列実行できるものと解釈することによってメモリアクセス命令が並列イネーブルフィールドを含まないように動作する、請求項１記載の処理エンジン。

【請求項１７】　メモリアクセス命令が、前記命令バッファ内の一対の命令の第１のプログラムされた命令として強制される、請求項１記載の処理エンジン。

【請求項１８】　キーボードアダプタを介して前記プロセッサに接続された一体型キーボードと、

ディスプレイアダプタを介して前記プロセッサに接続されたディスプレイと、

前記プロセッサに接続された無線周波数（ＲＦ）回路と、

該ＲＦ回路に接続されたアンテナと、

をさらに含む、請求項１記載のデジタルシステム。

【請求項１９】　実行用の命令を準備する命令前処理手段をさらに含み、

該命令前処理手段が、別々のプログラムされたメモリ命令を結合して複合メモリ命令を形成するように動作する、請求項１記載のデジタルシステム。

【請求項２０】　処理エンジンの性能改善方法であって、

別々のプログラムされたメモリ命令から形成される複合命令をバッファするステップであって、該複合命令が、所定の複合命令タグを含むタグフィールドを含む、ステップと、

前記命令バッファ内の命令の前記タグフィールド内の前記所定の複合命令タグに応答して、前記複合命令から、少なくとも第１のプログラムされた命令に対する第１の

データフロー制御および第２のプログラムされた命令に対する第２のデータフロー制御を復号するステップと、を含む、方法。

【請求項２１】　別々の第１および第２のプログラムされたメモリ命令を結合して前記複合命令を形成するステップをさらに含む、請求項２０記載の方法。

【請求項２２】　少なくとも前記複合命令の複合アドレスフィールドからの前記第１のプログラムされたメモリ命令に対する第１のメモリアドレスおよび前記第２のプログラムされたメモリ命令に対する第２のメモリアドレスを復号するステップをさらに含む、請求項２０記載の方法。

【請求項２３】　ハードプログラムされたデュアルメモリ命令に対する前記アドレスフィールドと同じビット位置からの前記複合命令の前記複合アドレスフィールドを復号するステップをさらに含む、請求項２２記載の方法。

【請求項２４】　前記複合命令の第１の命令に対する分割操作符号を復号するステップをさらに含む、請求項２０記載の方法。

【請求項２５】　前記複合命令の前記第１の命令に対する縮小サイズ操作符号を復号するステップをさらに含む、請求項２４記載の方法。

【請求項２６】　前記応答ステップが、前記第１および第２のプログラムされたメモリ命令のデータアドレス発生（ＤＡＧＥＮ）フィールドから形成される結合ＤＡＧＥＮフィールドを復号するステップを含む、請求項２１記載の方法。

【請求項２７】　前記結合ＤＡＧＥＮフィールドが、結合アドレスフィールドの一部を形成する、請求項２６記載の方法。

【請求項２８】　前記復号機構が、所定のＤＡＧＥＮタグに応答して、前記結合ＤＡＧＥＮフィールドを復号する、請求項２６記載の方法。

【請求項２９】　第１および第２のメモリアドレスからそれぞれ識別されるアドレスから第１および第２のオペランドを並列にフェッチするステップをさらに含む、請求項２２記載の方法。

【請求項３０】　前記複合命令の第１および第２のプログラムされた命令に対する第１および第２のデータフロー操作の結果を並列に書き込むステップを含む、請求項２９記載の方法。

【請求項３１】　前記結合ステップが、前記別々のプログラムされたメモリ命令を前記複合命令のアセンブリの前に結合できるかどうかを決定するステップを含む、請求項２１記載の方法。

【請求項３２】　前記結合ステップが、
プログラムされたメモリ命令が結合できることを決定するステップと、
該決定されたプログラムされたメモリ命令を結合して複合メモリ命令を形成するステップと、をさらに含む、請求項３１記載の方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、処理エンジンと、このような処理エンジンでの命令の並列実行に関する。

【０００２】

【従来の技術】多数の命令実行ユニットを使用してマイクロプロセッサで命令を並列実行することが知られている。このような並列実行を行うための多くの異なるアーキテクチュアが知られている。並列実行により全体処理速度が向上する。典型的には、多数の命令が並列に命令バッファに与えられたのち並列に復号されて、実行ユニットにディスパッチされる。マイクロプロセッサは、そこでソフトウェアを実行するために高い命令スループットを必要とする汎用処理エンジンであり、それは、関係する特定のソフトウェアアプリケーションに応じて広範な処理要求を有することがある。さらに、並列性をサポートするために、並列実行の命令のスケジューリングを制御するための複雑なオペレーティングシステムが必要とされている。

【０００３】多くの異なるタイプの処理エンジンが知られており、マイクロプロセッサは単なる一例にすぎない。たとえば、特に特定の応用に対してデジタル信号プロセッサ（ＤＳＰ）が広く知られている。ＤＳＰは、典型的には、関連するアプリケーションの性能を最適化するように構成され、それを達成するために、より特殊化された実行ユニットおよび命令セットを利用している。

【０００４】

【発明が解決しようとする課題】本発明は、限定はしないが、デジタル信号プロセッサのような処理エンジンの性能改善に向けられている。

【０００５】

【課題を解決するための手段】本発明の第１の態様によれば、単一の実行未決複合命令をバッファするように動作する命令バッファと、命令バッファからの命令を復号するように構成された復号機構とを含む処理エンジンが提供される。復号機構は、命令のタグフィールドの所定のタグに応答するように構成されており、所定のタグは、命令が別々のプログラムされたメモリ命令から形成される複合命令であることを表す。復号機構は、所定のタグに応答して、少なくとも第１のプログラムされた命令に対する第１のデータフロー制御および第２のプログラムされた命令に対する第２のデータフロー制御を復号するように動作する。

【０００６】したがって、本発明の実施例は、別々のプログラムされた命令を結合することによって形成される（たとえば、アセンブルまたはコンパイル）複合命令に応答する復号機構を提供する。このようにして、処理エンジン内で利用できる帯域幅の使用を最適化することが

できる。したがって、適切なメモリ命令のような適切なプログラムされた命令をアセンブルまたはコンパイルして複合命令を形成することができる。複合命令からプログラムされた命令の各成分に対する別々の制御フローを発生することによって、これらの命令を完全にまたは部分的に並列に実施して処理エンジンの全体スループットに好ましい影響を及ぼすことができる。プログラムされた各命令に対して復号機構によって発生される制御フローは、単一命令として命令バッファ内に保持されている場合には、プログラムされた命令に対して発生されるものと同じとすることができる。

【0007】コンパクトで効率的な符号化が本発明の実施例で可能とされる。たとえば、メモリ命令は必ず所定の複合命令の形の命令バッファ内の第1の一対の命令であることを保証することによって、メモリアクセス命令の並列性に効率的な符号化、リアルエステートの効率的使用および消費電力低減を与えることができる。

【0008】本発明の実施例では、複合命令は、別々のプログラムされたメモリ命令から（たとえば、コンパイラやアセンブラのような命令前処理機構を使用して）結合することによって形成されるソフト複合メモリ命令として定義される。特定の実施例では、複合命令はフソトデュアルメモリ命令、すなわち、別々の第1および第2のプログラムされたメモリ命令からアセンブルされたデュアルメモリ命令であるが、別の例では3つ以上の命令をアセンブルして複合命令とすることができる。

【0009】好ましくは、復号機構は、複合命令内の複合メモリアドレスからの第1のプログラムされたメモリアドレス命令に対する第1のメモリアドレスおよび第2のプログラムされたメモリ命令に対する第2のメモリアドレスを復号するように動作する。特に、複合命令の複合アドレスフィールドがハードプログラムされたデュアルメモリ命令に対するアドレスフィールドと同じビット位置であれば、これは命令スループットに好ましい影響を及ぼすことができる。この場合、アドレスの復号は、命令の操作符号がデュアル命令の第1および第2の命令のフォーマットに無関係に復号される前に開始することができる。

【0010】複合命令に対して必要なビット数を減少するために、複合命令の複合アドレスフィールド内のメモリアドレスは間接アドレスとして配列され、それによって、復号機構はそのような命令に対する間接アドレスを復号するように動作するだけでよい。デュアル命令は単一命令よりも少ないオプションをサポートするので、アドレスに対するポスト変更フィールドのサイズを縮小し、それによって、アドレス自体に必要なビット数を減少し、かつ、間接／直接インジケータビットを不要とすることができる。

【0011】メモリアクセス命令は、命令バッファ内の一対の命令の第1の命令として強制することができる。この場合、ソフトデュアル命令は、2つのメモリ命令に対応する符号化を有効に提供する。その結果、並列イネーブルフィールドの必要性は回避され、任意のメモリ命令を暗黙のうちに並列性とすることができる。さらに、それにより、外部インターフェイス帯域幅を最適化しかつキャッシュミスを低減しながらアプリケーション符号サイズを縮小する利点が得られる。

【0012】命令対の第2の命令に対するデコーダはまた、第1の命令に対するデコーダのサブセットとすることができ、必要な集積回路リアルエステートおよび処理エンジンの消費電力が低減される。

【0013】コンパクトな命令フォーマットを提供するとともに、アドレスフィールドをハード複合命令に対するものと同じ位置に配置できるようにするために、複合命令は、所定の複合命令の第1の命令に対する分割操作符号フィールドを含むことができる。たとえば、操作符号はアドレスフィールドのいずれか側で分割することができる。デコーダは、適切なタグフィールドの検出に応答して、複合命令の第1の命令に対する分割操作符号を復号することができる。

【0014】ビット数をさらに減少するために、複合命令は、所定の複合命令の少なくとも第1の命令に対する縮小操作符号フィールドを含み、操作符号フィールドが第1のプログラムされた命令の操作符号フィールドよりも少ないビットを含むようにすることができる。メモリ命令に対する操作符号の範囲をある範囲内に制約することによって、第1の操作符号に与える必要のあるビット数を減少することができる。復号機構は、所定のタグに応答して複合命令の第1の命令に対する縮小サイズ操作符号を復号するように構成することができる。

【0015】上述したさまざまな方策により、所定の複合命令は、別々のプログラムされた命令の総ビット数と同じ総ビット数を有するように構成することができる。プログラムされた命令からのフィールドの再構成によって、他の命令と共通の全体フォーマットを有する所定の複合命令が得られる。

【0016】プログラムされた各命令がデータアドレス発生（ＤＡＧＥＮ）コードフィールドを有する場合には、個別のプログラムされた命令の個別のＤＡＧＥＮコードを複合命令内の結合ＤＡＧＥＮコードフィールドに結合することができる。これにより、複合命令のより迅速な復号および実行を行うことができる。結合ＤＡＧＥＮコードフィールドは結合アドレスフィールドの一部を形成することができる。結合ＤＡＧＥＮコードフィールドが与えられる場合には、復号機構は所定のＤＡＧＥＮタグに応答して結合ＤＡＧＥＮフィールドを復号することができる。

【0017】処理エンジンには、第1および第2のメモリアドレスによってそれぞれ識別されるアドレスからの第1および第2のオペランドを並列にフェッチするよう

に動作するデータフェッチコントローラを設けることができる。データライトコントローラも、第1および第2の命令に対する第1および第2のデータフロー操作の結果を並列に書き込むように動作することができる。また、デュアル・リード／ライト操作を行うこともできる。

【0018】本発明の実施例では、アセンブラシンタクスは、ハード複合およびソフト複合シンタクス間を区別して、並列性に利用できるスロットに対する可視性を与えることができる。ハード複合命令は、バス／オペレータ資源競合がない限り並列イネーブルビットによって表示される制御フローやレジスタ命令のような非メモリ命令と並列に実行することができる。

【0019】本発明の他の態様によれば、上述した処理エンジンを含むプロセッサ、たとえば必ずしもそうである必要はないがデジタル信号プロセッサが提供される。プロセッサは、たとえば特定用途集積回路（ＡＳＩＣ）のような集積回路として実現することができる。

【0020】上述した処理エンジンを含むデジタル信号処理システムには、別々のプログラムされたメモリ命令を結合して複合メモリ命令を形成するように動作する命令前処理機構を設けることもできる。命令プリプロセッサはコンパイラやアセンブラなどとすることができ、それはプログラムされた命令からの複合命令をコンパイルまたはアセンブルするように動作する。この機構は、複合命令のアセンブリの前に別々のプログラムされたメモリ命令を結合できるかを確認するように構成することができる。

【0021】本発明のさらに他の態様によれば、デジタル信号処理システム用の命令プリプロセッサが提供され、それは、結合することができるプログラムされたメモリ命令を確認し、前記確認されたプログラムされたメモリ命令からの複合メモリ命令をアセンブルするように構成される。

【0022】ここで、「命令プリプロセッサ」という用語は、コンパイラやアセンブラなどを含む、命令を処理するすなわち命令をコンパイルおよび／またはアセンブルする任意の機構を含むものと広く解釈すべきことを理解願いたい。

【0023】命令プリプロセッサは、たとえばデータ記憶媒体（ディスク，固体メモリ，電気や光学や他の電磁気のようなデータ伝送媒体（たとえば、ワイヤレス伝送媒体））のような搬送媒体上に別々に設けることができる。

【0024】本発明のその他の態様によれば、処理エンジンの性能改善方法が提供される。この方法は、別々のプログラムされたメモリ命令からアセンブルされた複合命令をバッファするステップであって、複合命令が所定の複合命令タグを含むタグフィールドを含む、ステップと、命令バッファ内の命令のタグフィールド内の所定の複合命令タグに応答して、複合命令から、少なくとも第1のプログラムされた命令に対する第1のデータフロー制御および第2のプログラムされた命令に対する第2のデータフロー制御を復号するステップと、を含んでいる。

【0025】

【発明の実施の形態】本発明は、たとえば特定用途集積回路（ＡＳＩＣ）内に実現されるデジタル信号プロセッサ（ＤＳＰ）に特に応用されるが、他の形式の処理エンジンにも応用される。

【0026】図1は、本発明の一実施例を有するマイクロプロセッサ１０のブロック図である。マイクロプロセッサ１０は、デジタル信号プロセッサ（ＤＳＰ）である。分かり易くするために、図1は、マイクロプロセッサ１０の本発明の一実施例を理解するのに関係のある部分のみを示す。ＤＳＰの一般的構造の詳細は、よく知られており、他で容易に確かめることができる。たとえば、フレデリック・ブートウドらの米国特許第５，０７２，４１８号には、ＤＳＰが詳細に記載されており、本開示の一部としてここに援用する。ギャリー・スオボダらの米国特許第５，３２９，４７１号には、ＤＳＰのテストおよびエミュレート方法が詳細に記載されており、本開示の一部としてここに援用する。マイクロプロセッサの分野の当業者であれば本発明を製造し使用できるように、マイクロプロセッサ１０の本発明の一実施例に関連する部分の詳細が、以下に十分詳しく説明される。

【0027】本発明の態様から利益を得ることができるいくつかのシステムの例が、本開示の一部としてここに援用される米国特許第５，０７２，４１８号に、特に米国特許第５，０７２，４１８号の図２～図１８に記載されている。性能を改善するかコストを低減する本発明の一態様を組み入れたマイクロプロセッサを使用して、米国特許第５，０７２，４１８号に記載されたシステムをさらに改善することができる。そのようなシステムは、限定はしないが、産業プロセスコントロール，自動車システム，モータコントロール，ロボットコントロールシステム，衛星電気通信システム，エコーキャンセリングシステム，モデム，ビデオイメージングシステム，音声認識システムおよび暗号付ボコーダーモデムシステムなどを含む。

【0028】図1のマイクロプロセッサのさまざまなアーキテクチュア上の特徴および完全な命令セットの説明が、同じ譲受人による特許出願第９８４０２４５５．４号（ＴＩ－２８４３３）に記載されており、本開示の一部としてここに援用する。

【0029】次に、本発明によるプロセッサの一例の基本的アーキテクチュアについて説明する。図1は、本発明の一つの典型的な実施例を形成するプロセッサ１０の全体略図である。プロセッサ１０は、処理エンジン１００とプロセッサバックプレーン２０とを含んでいる。本

実施例では、プロセッサは、特定用途集積回路（ＡＳＩＣ）に実現されたデジタル信号プロセッサ１０である。

【００３０】図１に示すように、処理エンジン１００は、処理コア１０２と処理コア１０２を処理コア１０２の外部のメモリユニットとインターフェイスさせるメモリインターフェイスすなわち管理ユニット１０４とを有する中央処理装置（ＣＰＵ）を形成する。

【００３１】プロセッサバックプレーン２０は、バックプレーンバス２２を含み、それには処理エンジンのメモリ管理ユニット１０４が接続されている。バックプレーンバス２２には、命令キャッシュメモリ２４，周辺装置２６および外部インターフェイス２８も接続されている。

【００３２】他の実施例では、異なる構成および／または異なる技術を使用して本発明を実現できることが分かるであろう。たとえば、処理エンジン１００はプロセッサ１０を形成することができ、プロセッサバックプレーン２０はそこから分離されている。処理エンジン１００は、たとえば、バックプレーンバス２２，周辺装置および外部インターフェイスを支持するバックプレーン２０から独立してその上に搭載されたＤＳＰであり得る。処理エンジン１００は、たとえば、ＤＳＰではなくマイクロプロセッサとすることができ、ＡＳＩＣ技術以外の技術で実現することができる。処理エンジンまたは処理エンジンを含むプロセッサは１つ以上の集積回路に実現することができる。

【００３３】図２は、処理コア１０２の一実施例の基本構造を示す。図から分かるように、処理コア１０２は、４つの要素、すなわち、命令バッファユニット（Ｉユニット）１０６と３つの実行ユニットとを含んでいる。実行ユニットは、プログラムフローユニット（Ｐユニット）１０８と、アドレスデータフローユニット（Ａユニット）１１０と、命令バッファユニット（Ｉユニット）１０６から復号された命令を実行しプログラムフローを制御かつ監視するデータ計算ユニット（Ｄユニット）１１２とである。

【００３４】図３は、処理コア１０２のＰユニット１０８，Ａユニット１１０およびＤユニット１１２を詳細に示すとともに、処理コア１０２のさまざまな要素を接続するバス構造を示す。Ｐユニット１０８は、たとえば、ループ制御回路と、ＧｏＴｏ／分岐制御回路と、リピートカウンタレジスタおよび割込みマスク，フラグまたはベクトルレジスタのようなプログラムフローを制御し監視するさまざまなレジスタとを含んでいる。Ｐユニット１０８は、汎用データライトバス（ＥＢ，ＦＢ）１３０，１３２とデータリードバス（ＣＢ，ＤＢ）１３４，１３６とアドレス定数バス（ＫＡＢ）１４２とに結合されている。さらに、Ｐユニット１０８は、ＣＳＲ，ＡＣＢおよびＲＧＤとラベルされたさまざまなバスを介してＡユニット１１０およびＤユニット１１２内のサブユニットに結合されている。

【００３５】図３に示すように、本実施例では、Ａユニット１１０はレジスタファイル３０とデータアドレス発生サブユニット（ＤＡＧＥＮ）３２と算術および論理演算装置（ＡＬＵ）３４とを含んでいる。Ａユニットレジスタファイル３０はさまざまなレジスタを含み、それら中には、アドレス発生だけでなくデータフローにも使用できる１６ビットポインタレジスタ（ＡＲ０，．．．，ＡＲ７）およびデータレジスタ（ＤＲ０，．．．，ＤＲ３）がある。さらに、レジスタファイルは、１６ビット巡回バッファレジスタと７ビットデータページレジスタとを含んでいる。汎用バス（ＥＢ，ＦＢ，ＣＢ，ＤＢ）１３０，１３２，１３４，１３６だけでなく、データ定数バス１４０およびアドレス定数バス１４２がＡユニットレジスタファイル３０に結合されている。Ａユニットレジスタファイル３０は、それぞれ反対方向に作動する１方向性バス１４４，１４６によってＡユニットＤＡＧＥＮユニット３２に結合されている。ＤＡＧＥＮユニット３２は、１６ビットＸ／Ｙレジスタと、たとえば処理エンジン１００内のアドレス発生を制御し監視する係数およびスタックポインタレジスタとを含んでいる。

【００３６】Ａユニット１１０は、加算，減算およびＡＮＤ，ＯＲおよびＸＯＲ論理演算子などのＡＬＵに典型的に関連する機能だけでなくシフタ機能も含むＡＬＵ３４も含んでいる。ＡＬＵ３４は、汎用バス（ＥＢ，ＤＢ）１３０，１３６および命令定数データバス（ＫＤＢ）１４０にも結合されている。ＡユニットＡＬＵは、Ｐユニット１０８レジスタファイルからレジスタ内容を受信するＰＤＡバスによってＰユニット１０８に結合されている。ＡＬＵ３４は、アドレスおよびデータレジスタ内容を受信するバスＲＧＡ，ＲＧＢとレジスタファイル３０のアドレスおよびデータレジスタに転送するバスＲＧＤとによってＡユニットレジスタファイル３０にも結合されている。

【００３７】図から分かるように、Ｄユニット１１２は、Ｄユニットレジスタファイル３６と、ＤユニットＡＬＵ３８と、Ｄユニットシフタ４０と、２つの乗算および累算ユニット（ＭＡＣ１，ＭＡＣ２）４２，４４とを含んでいる。Ｄユニットレジスタファイル３６とＤユニットＡＬＵ３８とＤユニットシフタ４０とは、バス（ＥＢ，ＦＢ，ＣＢ，ＤＢ，ＫＤＢ）１３０，１３２，１３４，１３６，１４０に結合され、また、ＭＡＣユニット４２，４４は、バス（ＣＢ，ＤＢ，ＫＤＢ）１３４，１３６，１４０とデータリードバス（ＢＢ）１４４とに結合されている。Ｄユニットレジスタファイル３６は、４０ビット累算器（ＡＣ０，．．．，ＡＣ３）と１６ビット遷移レジスタとを含んでいる。また、Ｄユニット１１２は、Ａユニット１１０の１６ビットポインタおよびデータレジスタをソースとして利用したり、４０ビット累算器の他にデスティネーションレジスタを利用すること

ができる。Dユニットレジスタファイル３６は、累算器ライトバス（ＡＣＷ０，ＡＣＷ１）１４６，１４８を介してDユニットＡＬＵ３８およびＭＡＣ１＆２　４２，４４から、また、累算器ライトバス（ＡＣＷ１）１４８を介してDユニットシフタ４０から、データを受信する。データは、累算器リードバス（ＡＣＲ０，ＡＣＲ１）１５０，１５２を介してDユニットレジスタファイル累算器からDユニットＡＬＵ３８，Dユニットシフタ４０およびＭＡＣ１＆２　４２，４４に読み出される。DユニットＡＬＵ３８とDユニットシフタ４０とは、ＥＦＣ，ＤＲＢ，ＤＲ２およびＡＣＢとラベルされたさまざまなバスを介してＡユニット１０８のサブユニットにも結合されている。

【００３８】図４を参照すると、３２ワード命令バッファキュー（ＩＢＱ）５０２を含む命令バッファユニット１０６が示されている。ＩＢＱ５０２は、８ビットバイト５０６に論理的に分割された３２×１６ビットレジスタ５０４を含んでいる。命令は、３２ビットプログラムバス（ＰＢ）１２２を介してＩＢＱ５０２に到来する。命令は、ローカルライトプログラムカウンタ（ＬＷＰＣ）５３２によって指示される位置に３２ビットサイクルでフェッチされる。ＬＷＰＣ５３２は、Ｐユニット１０８に位置されたレジスタに含まれている。Ｐユニット１０８は、ローカルリードプログラムカウンタ（ＬＲＰＣ）５３６レジスタとライトプログラムカウンタ（ＷＰＣ）５３０レジスタおよびリードプログラムカウンタ（ＲＰＣ）５３４レジスタとをも含んでいる。ＬＲＰＣ５３６は、命令デコーダ５１２，５１４にロードされる次の一つまたは複数の命令のＩＢＱ５０２内の位置を指示する。すなわち、ＬＲＰＣ５３４は、デコーダ５１２，５１４に現在ディスパッチされている命令のＩＢＱ５０２内の位置を指示する。ＷＰＣは、パイプラインに対する命令コードの次の４バイトの始まりのプログラムメモリ内のアドレスを指示する。ＩＢＱ内への各フェッチに対して、プログラムメモリからの次の４バイトが命令境界とは無関係にフェッチされる。ＲＰＣ５３４は、デコーダ５１２，５１４に現在ディスパッチされている命令のプログラムメモリ内のアドレスを指示する。

【００３９】命令は、４８ビットワードに形成され、マルチプレクサ５２０，５２１を介して４８ビットバス５１６によって命令デコーダ５１２，５１４にロードされる。当業者ならば、命令は４８ビット以外のワードに形成することができること、また、本発明は前記した特定の実施例に限定されるものではないことが、分かるであろう。

【００４０】バス５１６は、任意の１命令サイクル中に、デコーダ当たり１つずつ、最大２つの命令をロードすることができる。命令の組合せは、４８ビットバスの両端間にわたって適合する８，１６，２４，３２，４０および４８ビットのフォーマットの任意の組合せとすることができる。１サイクル中に１命令しかロードできない場合には、デコーダ１，５１２がデコーダ２，５１４に優先してロードされる。次に、各命令は、それらを実行するために、また、命令または演算が実行されるべきデータにアクセスするために、各機能ユニットに転送される。命令デコーダに通される前に、命令はバイト境界上でアラインされる。アライメントは、その復号中に前の命令に対して引き出されたフォーマットに基づいて行われる。バイト境界を有する命令のアライメントに関連する多重化は、マルチプレクサ５２０，５２１で実行される。

【００４１】プロセッサコア１０２は7ステージパイプラインを介して命令を実行し、その各ステージは図５を参照して説明される。

【００４２】パイプラインの第１ステージは、ＰＲＥ－ＦＥＴＣＨ（Ｐ０）ステージ２０２であり、このステージ中に、メモリインターフェイスまたはメモリ管理ユニット１０４のアドレスバス（ＰＡＢ）１１８上にアドレスを表明することによって次のプログラムメモリ位置がアドレス指定される。

【００４３】次のステージ、ＦＥＴＣＨ（Ｐ１）ステージ２０４では、プログラムメモリが読み出され、Ｉユニット１０６がメモリ管理ユニット１０４からＰＢバス１２２を介して充填される。

【００４４】パイプラインはＰＲＥ－ＦＥＴＣＨおよびＦＥＴＣＨステージ中に割り込まれて逐次プログラムフローを中断してプログラムメモリ内の他の命令、たとえば分岐命令を指示することができる点で、ＰＲＥ－ＦＥＴＣＨおよびＦＥＴＣＨステージは残りのパイプラインステージから独立している。

【００４５】次に、命令バッファ内の次の命令が、第３ステージＤＥＣＯＤＥ（Ｐ２）２０６でデコーダ５１２または複数のデコーダ５１４にディスパッチされ、そこで、命令は、復号されて、その命令を実行する実行ユニット、たとえばＰユニット１０８，Ａユニット１１０またはDユニット１１２にディスパッチされる。復号ステージ２０６は、命令のクラスを示す第１の部分と命令のフォーマットを示す第２の部分と命令に対するアドレス指定モードを示す第３の部分とを含む命令の少なくとも一部を復号することを含んでいる。

【００４６】次のステージはＡＤＤＲＥＳＳ（Ｐ３）ステージ２０８であり、そこでは、命令内で使用されるデータのアドレスが計算されるか、命令がプログラム分岐すなわちジャンプを必要とする場合には新しいプログラムアドレスが計算される。各計算は、Ａユニット１１０またはＰユニット１０８でそれぞれ行われる。

【００４７】ＡＣＣＥＳＳ（Ｐ４）ステージ２１０では、リードオペランドのアドレスが出力されたのち、Ｘｍｅｍ間接アドレス指定モードを有するＤＡＧＥＮ Ｘ演算子でアドレスが発生されているメモリオペランド

が、間接アドレス指定されたXメモリ（Ｘｍｅｍ）から読み出される。

【００４８】パイプラインの次のステージはＲＥＡＤ（Ｐ５）ステージ２１２であり、そこでは、Ｙｍｅｍ間接アドレス指定モードを有するＤＡＧＥＮ Ｙ演算子内または係数アドレスモードを有するＤＡＧＥＮ Ｃ演算子内でアドレスが発生されているメモリオペランドが、読み出される。命令の結果が書き込まれるメモリ位置のアドレスが出力される。

【００４９】デュアルアクセスの場合には、リードオペランドをYパスで発生し、ライトオペランドをXパスで発生することもできる。

【００５０】最後に、命令がAユニット１１０内またはDユニット１１２内で実行される実行ＥＸＥＣ（Ｐ６）ステージ２１４がある。次に、結果がデータレジスタまたは累算器に格納されるか、リード／モディファイ／ライト用またはストア命令用のメモリに書き込まれる。さらに、シフト演算がＥＸＥＣステージ中に累算器内のデータになされる。

【００５１】次に、パイプラインプロセッサの動作の基本的原理について図６を参照して説明する。図６から分かるように、第１の命令３０２に対して、連続パイプラインステージが期間T_1～T_7にわたって行われる。各期間はプロセッサマシンクロックに対するクロックサイクルである。前の命令が次のパイプラインステージに移行しているため、第２の命令３０４が期間T_2でパイプラインに入ることができる。第３の命令３０６に対して、ＰＲＥ－ＦＥＴＣＨステージ２０２が期間T_3で行われる。図６から分かるように、７ステージパイプラインに対して、合計７つの命令を同時に処理することができる。７つの命令３０２～３１４の全てに対して、図６は期間T_7でそれら全てが処理中であることを示している。このような構造は命令の処理に一形式の並列性を付加する。

【００５２】図７に示すように、本発明のこの実施例は、２４ビットアドレスバス１１４および双方向１６ビットデータバス１１６を介して外部メモリユニット（不図示）に結合されるメモリ管理ユニット１０４を含んでいる。さらに、メモリ管理ユニット１０４は２４ビットアドレスバス１１８および３２ビット双方向データバス１２０を介してプログラム格納メモリ（不図示）に結合されている。メモリ管理ユニット１０４は３２ビットプログラムリードバス（ＰＢ）１２２を介してマシンプロセッサコア１０２のＩユニット１０６にも結合されている。Ｐユニット１０８，Ａユニット１１０およびＤユニット１１２はデータリードおよびデータライトバスおよび対応するアドレスバスを介してメモリ管理ユニット１０４に結合されている。Ｐユニット１０８はさらにプログラムアドレスバス１２８に結合されている。

【００５３】より詳細には、Ｐユニット１０８は２４ビットプログラムアドレスバス１２８と２つの１６ビットデータライトバス（ＥＢ，ＦＢ）１３０，１３２と２つの１６ビットデータリードバス（ＣＢ，ＤＢ）１３４，１３６とによってメモリ管理ユニット１０４に結合されている。Ａユニット１１０は、２つの２４ビットデータライトアドレスバス（ＥＡＢ，ＦＡＢ）１６０，１６２と２つの１６ビットデータライトバス（ＥＢ，ＦＢ）１３０，１３２と３つのデータリードアドレスバス（ＢＡＢ，ＣＡＢ，ＤＡＢ）１６４，１６６，１６８と２つの１６ビットデータリードバス（ＣＢ，ＤＢ）１３４，１３６とを介してメモリ管理ユニット１０４に結合されている。Ｄユニット１１２は、２つのデータライトバス（ＥＢ，ＦＢ）１３０，１３２と３つのデータリードバス（ＢＢ，ＣＢ，ＤＢ）１４４，１３４，１３６とを介してメモリ管理ユニット１０４に結合されている。

【００５４】図７は、たとえば分岐命令を転送する、Ｉユニット１０６からＰユニット１０８への命令の通過を参照符号１２４で表示している。さらに、図７は、Ｉユニット１０６からＡユニット１１０およびＤユニット１１２へのデータの通過を参照符号１２６，１２８でそれぞれ表示している。

【００５５】本発明のこの実施例では、処理エンジン１００はいくつかのフォーマットでマシン命令に応答する。さまざまなフォーマットのこのような命令の例を以下に示す。

【００５６】８ビット命令：ＯＯＯＯ ＯＯＯＯ
これは、８ビット命令、たとえばメモリマップ修飾子（ＭＭＡＰ()）またはリードポート修飾子（readport()）を表す。このような修飾子は単に８ビット操作符号（ＯＯＯＯ ＯＯＯＯ）を含むのみである。このような場合、並列性はインプリシットである。

【００５７】１６ビット命令：ＯＯＯＯ ＯＯＯＥ ＦＳＳＳ ＦＤＤＤ
これは、１６ビット命令、たとえばデスティネーションレジスタの内容（たとえば、ｄｓｔ）がそのレジスタの前の内容（ｄｓｔ）とソースレジスタの内容（ｓｒｃ）との和となる命令、すなわち、

【００５８】

【数１】ｄｓｔ＝ｄｓｔ＋ｓｒｃ

【００５９】の一例を表わす。

【００６０】このような命令は、１ビットパラレルイネーブルフィールド（Ｅ）と４ビットソースレジスタ識別子（ＦＳＳＳ）と４ビットデスティネーションレジスタ識別子（ＦＤＤＤ）とを有する７ビット操作符号（ＯＯＯＯ ＯＯＯ）である。

【００６１】１６ビット命令：ＯＯＯＯ ＦＤＤＤ ＰＰＰＭ ＭＭＭＩ
これは、たとえばデスティネーションレジスタの内容（たとえば、ｄｓｔ）がメモリ位置の内容（Ｓｍｅｍ）となる、すなわち、

【0062】
【数2】dst=Smem
【0063】16ビット命令のもう1つの例である。
【0064】このような命令は、4ビット操作符号（OOOO）と4ビットデスティネーションレジスタ識別子（FDDD）と3ビットポインタアドレス（PPP）と4ビットアドレス変更子（M MMM）と直接/間接アドレスインジケータ（I）とを含んでいる。
【0065】24ビット命令：OOOO OOOE LLLL LLLL oCCC CCCC
これは、24ビット命令、たとえば条件分岐命令および条件が満たされる場合のオフセット（L8）を表す、すなわち、
【0066】
【数3】if（cond） goto L8
【0067】の一例を表わす。
【0068】このような命令は、1ビットパラレルイネーブルフィールド（E）と8ビット分岐オフセット（LLLL LLLL）と1ビット操作符号拡張（o）と7ビット条件フィールド（CCC CCCC）とを有する7ビット操作符号（OOOOOOO）を含んでいる。
【0069】24ビット命令：OOOO OOOO PPPM MMMI SSDD ooU%
これは、24ビット命令のもう1つの例、たとえば累算器の内容（ACy）がもう1つの累算器の内容（ACx）およびメモリ位置の内容（随意丸めがある）の二乗の和を丸めた結果となり、データレジスタの内容（DR3）が随意メモリ位置の内容となる単一メモリオペランド命令、すなわち、
【0070】
【数4】ACy=rnd（ACx*Smem*Smem），DR3=Smem
【0071】のもう1つの例である。
【0072】このような命令は、8ビット操作符号（OOOO OOOO）と3ビットポインタアドレス（PPP）と4ビットアドレス変更子（M MMM）と1ビット直接/間接アドレスインジケータフィールド（I）と2ビットソース累算器識別子（SS）と2ビットデスティネーション累算器識別子（DD）と2ビット操作符号拡張（oo）と更新条件フィールド（u）と1ビット丸めオプションフィールド（%）とを含んでいる。
【0073】32ビット命令：OOOO OOOO PPPM MMMI KKKK KKKKKKKK KKKK
これは、32ビット命令、たとえばメモリ位置（Smem）の一定値（K16）との符号比較に応じてテストレジスタの内容（TC1）が1または0に設定される命令、すなわち、
【0074】
【数5】TC1=（Smem==K16）
【0075】の一例である。
【0076】このような命令は、8ビット操作符号（OOOO OOOO）と3ビットポインタアドレス（PPP）と4ビットアドレス変更子（M MMM）と1ビット直接/間接アドレスインジケータフィールド（I）と16ビット定数フィールド（KKKK KKKK KKKK KKKK）とを含んでいる。
【0077】ハードデュアル命令：OOOO OOOO XXXM MMYY YMMM SSDD ooox ssU%
これは、「ハードデュアルアクセス命令」と呼ぶことができる32ビットデュアルアクセス命令、または、たとえばプログラマによってのようにプログラムされたデュアル命令であるハードプログラムされたデュアルメモリ命令である。このような命令は2つのDAGEN演算子を必要とする。第2の命令は並列に実行することができる。それは、典型的には、レジスタまたは制御命令である。バス競合がないかぎり、メモリスタック命令も並列に実行することができる。このような命令の一例は次のようである。
【0078】
【数6】Cy=rnd（DRx*Xmem），Ymem=HI（ACx<<DR2）
DR3=Xmem
【0079】この命令は、8ビット操作符号（OOOO OOOO），4ビットアドレス変更子（M MMM）付き3ビットXmemポインタアドレス（XXX），4ビットアドレス変更子（M MMM）付き3ビットYmemポインタアドレス（YYY），2ビットソースアキュムレータ（ACx）識別子（SS），2ビットデスティネーションアキュムレータ（ACy）識別子（DD），3ビット操作符号拡張（ooo），ドントケアビット（x），2ビットソースアキュムレータ識別子（ss），1ビットオプショナルDR3更新フィールド（U）および1ビットオプショナル丸めフィールド（%）を含んでいる。
【0080】図8は、命令対およびソフトデュアル命令を形成する命令の組合せを示す表である。このような命令対では、対の第1の命令は常にメモリ操作である。第2の命令もメモリ命令である場合、それはソフトデュアル命令すなわち複合命令として構成されることが分かるであろう。
【0081】命令対の第2の位置に（すなわち、対のより高いプログラムアドレスに対して）配置される命令は、一対の命令の第1の命令と並列に命令を実施できるかどうかを示す並列イネーブルフィールド（Eビット）を含んでいる。並列イネーブルビットは、命令間の命令フォーマット境界から所定のオフセットで配置される。デコーダは、命令実行を制御するために「E」ビットに応答するように構成される。
【0082】命令対で最初にメモリ操作をさせる理由

は、プロセッサパイプラインのアドレス復号ステージに入るときに、デコーダは、命令のフォーマットを知らず、フォーマット境界がどこであるかさえも知らないためである。メモリアドレス復号化は、良好な命令スループットを保証するパイプラインのクリティカルステージの1つである。したがって、命令の正確な性質が確認される前であっても復号を開始できるようにするためには、復号されるメモリ命令に対するアドレスビットの位置およびサイズを確実に知る必要がある。

【００８３】メモリ命令が第１の命令として命令対内に配置されるように強制することにより生じるもう１つの利点は、並列演算が許可されるかどうかを示すフィールドをメモリ命令に含める必要がないことである。そのため、命令セットはより効率的となり符号サイズを改善することができる。

【００８４】他のもう１つの利点は、命令対の第２の命令を復号するのに必要なハードウェアは、命令対の第１の命令を復号するためのハードウェアのサブセットであるしか必要としないことである。第１の命令は、命令対の第２の命令よりも低いプログラムアドレスを有する命令対の命令である。したがって、命令対の高いプログラムアドレスを有する命令用の復号ハードウェアは、命令対の低いプログラムアドレスを有する命令用の復号ハードウェアのサブセットとすることができる。それにより、復号ハードウェアの実現および動作に必要なシリコン面積および消費電力を低減することができる。

【００８５】命令対の２つの命令を並列処理できる場合には、それは各復号および実行ステージで行われる。しかしながら、物理的なバスタイミング制約により、バス転送はふらつくことがある。

【００８６】図９は、デュアル命令を含むさまざまなタイプの命令に対するメモリアクセスが行われるパイプラインステージを示す。図４と同様に、図示するパイプラインステージは単なる説明用にすぎないことに留意すべきである。実際上、プリフェッチおよびフェッチステージは残りのステージから独立したフローを形成する。

【００８７】図９を図５と比べると、Ｐ１はフェッチステージ、Ｐ２は復号ステージ、Ｐ３はアドレス計算ステージ、Ｐ４はアクセスステージ、Ｐ５はリードステージ、Ｐ６は実行ステージを表す。Ｂは、Ｂバスを介したレジスタからの係数リードアクセスを表す。ＣおよびＤは、ＣおよびＤバスを介したメモリリードアクセスをそれぞれ表す。ＥおよびＦは、ＥおよびＦバスを介したライトアクセスをそれぞれ表す。パイプライン上にバブル（すなわち、ストール）を生じることなくリードおよびライトアクセスを所要サイクルで実行できるようにするために、復号はできるだけ早期に行われる。

【００８８】図１０は、デュアルメモリアクセス命令の特定の形式を示す。それは、並列性を含んでいる２つの併合されプログラムされた命令から有効に形成される。図１０のデュアルメモリ命令は、ソフトデュアル命令と呼ばれ、ここでは複合命令とも呼ばれる。それは、２つのプログラムされたシングルメモリアクセス命令をたとえばコンパイラやアセンブラで命令プロセッサ内で結合して形成される。すなわち、この複合命令は、デュアル命令のようにプログラマによってプログラムされたりプリプログラムされることはない。この形式の複合命令が提供されると、両方の命令が同じサイクルで実行される並列演算によりメモリアクセス性能を改善することができる。下記の特定の例では、ソフトデュアル命令は、デュアル変更子オプションを有する間接アドレッシングに制限される。その結果、結合した命令サイズに関するサイズペナルティなしに並列演算により性能向上を達成するようにソフトデュアル命令を符号化することができる。

【００８９】ソフトデュアル命令は５ビットタグフィールド７０１によって限定され、図１０に示すように個別の下記の命令フィールドが構成されている。タグフィールドのサイズは、特定のインプリメンテーションに関する制約の結果として生じる。すなわち、

－　全体符号化フォーマットは、２つのプログラムされた構成命令の符号化フォーマットの和よりも大きくならないように制約される。

－　全体命令フォーマットサイズは８の倍数である。

－　他のシングル命令に対する操作符号のアベイラビリティ。

【００９０】下記のものがタグフィールド７０１に続く。

－　第１の命令に対する操作符号フィールドの部分７０２。

－　第１の命令に対する間接メモリアドレス（ＸＸＸＭＭＭ）７０３および第２の命令に対する間接メモリアドレス（ＹＹＹＭＭＭ）７０４を含む複合アドレスフィールド７０３／７０４。

－　第１の命令に対する操作符号フィールド７０５の残部。

－　第１の命令に対するデータフローフィールド７０６。

－　第２の命令の操作符号に対する操作符号フィールド７０７。

－　第２の命令に対するデータフローフィールド７０８。

【００９１】したがって、ソフトデュアル命令に対する結合アドレス部は、他の任意のデュアル命令に対するものとソフトデュアル命令内の同じ位置に保持される。それにより、関連する命令タイプを知ることなくアドレス復号を開始できる結果、高速アドレス復号の利点が得られる。それを達成するために、上述したように、ソフトデュアル命令内のビットを幾分再構成する必要があることが分かるであろう。

【００９２】２つのプログラムされた命令の各々がデータアドレス発生（ＤＡＧＥＮ）フィールドを含む上述した修正に加えて、それらを結合してソフトデュアル命令内に結合ＤＡＧＥＮフィールドを形成することができる。結合ＤＡＧＥＮフィールドを設けることにより、ソフトデュアル命令の後の実行を容易にし速度を速めることができる。

【００９３】図１１は、２つの独立命令をソフトデュアル命令に変換するためのさまざまなステップを示す。２つの独立命令７２１，７２２はステージ７２０に表示されている。

【００９４】ステージ７２３で示すように、最初の２４ビット命令７２１は、第１バイト内の８ビット操作符号７２４と、次のバイト内のシングルメモリ（Ｓｍｅｍ）アドレス７２５と、次のバイト内のデータフロービット７２６とを含んでいる。第２の２４ビット命令７２２は、第１バイト内の８ビット操作符号７２７と、次のバイト内のシングルメモリアドレス７２８と、次のバイト内のデータフロービット７２９とを含んでいる。ステージ７３０において、８操作符号ビットはそれぞれ、各命令の操作符号バイト７２４，７２７内で「Ｏ」とラベルされている。シングルメモリアドレス７２５，７２８はそれぞれ、７アドレスビット「Ａ」＋間接／直接インジケータビット「Ｉ」を含むように示されている。それは、標準メモリアクセスに対するアドレスが直接または間接となることがあるためである。図示する例では、粒度はバイトに基づいている。しかしながら、他の例では、８ビット以外に基づいた粒度を利用することができる。さらに、２つの命令は対称的とする必要はなく、第１の命令は第２の命令とは異なるバイト数とすることができる。

【００９５】ステージ７３５において、第１の命令の操作符号７２４は２つの部分に分割される。操作符号７２４の８ビットのうち７ビットだけを考えればよい。それは、（たとえば、ソフトデュアル命令に対して１６進法でたとえば８０～ＦＦの所定範囲内に全てのメモリ命令が操作符号を有することを保証することにより）ソフトデュアル命令の場合にそれが冗長であることを保証することができるメモリコードマッピングの結果である。後でステージ７２６，７４０および図１０で分かるように、最初の命令に対する操作符号は分割される。最初の命令に対する操作符号の３ビットがソフトデュアル命令タグ７３７と第１および第２の命令に対する結合アドレス７３８との間に配置され、４ビットが結合アドレス７３８の後に配置される。

【００９６】ステージ７３６では、ソフトデュアル命令タグ７３７の挿入が示されている。これは、ソフトデュアル命令を表すものとしてデコーダによって解釈されることがあるタグである。シングルメモリフィールド７２５，７２８の併合も図示されている。これは全ての命令が間接アドレスに制限されるために達成することができ、それにより、間接／直接フラグが不要である。間接アドレスは、第１および第２の命令に対する３ビットベースアドレスＸＸＸまたはＹＹＹと３ビット変更子（ＭＭＭ）とによってそれぞれ表示される。ステージ７３６は第２の命令の第１のバイト位置への第１の命令に対するデータフローの移動を示し、第２の命令に対する操作符号はその命令の第２のバイト位置に移動される。

【００９７】その結果、図１０に示すソフトデュアル命令のフォーマットが達成される。ソフトデュアル命令対２つのシングルメモリアクセス命令に対する符号サイズペナルティがないことが分かる。２つのシングルメモリ（Ｓｍｅｍ）命令をＸｍｅｍ，Ｙｍｅｍで置換することによって、「ソフトデュアル」タグ７０１／７３７を挿入するのに十分なビットが解放される。ソフトデュアルタグ自体により、デコーダは命令対をメモリ命令として復号すべきことを検出することができる。命令セットマッピングを使用してメモリ命令がウィンドウ８０～ＦＦ内で符号化されることを保証することができ、それによって、第１の操作符号７２４の最上位ビット（ビット７）をデュアルフィールド符号化を遂行するときに廃棄することができる。

【００９８】図示する例では、図１１に示したさまざまなステージは、実行する命令を準備するときに、命令プロセッサ，たとえばコンパイラまたはアセンブラによって実施される。命令プロセッサによって行われるステップは図１２にフロー図で示されている。

【００９９】ステップＳ１において、命令プロセッサは、ソフトデュアル命令に結合される可能性のある２つの命令の存在を検出する。それを可能とするために、命令は並列に行うことができかつデータまたはコントロールフロー不整合を生じないものとする必要がある。命令セット内の各命令は、アドレスジェネレータリソースと命令をサポートするのに関連したメモリアクセスのタイプとを定義するＤＡＧＥＮタグ内のＤＡＧＥＮ変数によって限定される。

【０１００】したがって、ステップＳ２において、命令プロセッサは、ＤＡＧＥＮ変数を解析することによって２つのスタンドアロンメモリ命令を併合してソフトデュアル命令とすることの実行可能性を決定する第１のステップを行う。これがチェックアウトされるものとすると、命令プリプロセッサは、潜在的バスおよびオペレータ競合を解析し、第１および第２の命令の結合に潜在的なバーがあるかどうかを立証するように作動することができる。

【０１０１】ステップＳ３において、命令プリプロセッサは、ソフトデュアル命令タグ７３７を適用し、図１１に示すフィールド位置だけでなく操作符号およびアドレス表示も修正する。ステップＳ４において、命令プリプロセッサによってソフトデュアル命令が出力される。

【0102】図13は、ソフトデュアル命令に対する復号プロセスを示す略ブロック図である。図13は、命令バッファユニット106からの48ビット命令ワード800の復号を示す。

【0103】図13に示すように命令ワードの左に配置される操作符号（opcode）から、操作符号復号回路の論理802，804は、組込みデュアルまたはソフトデュアル命令が復号されるべきかどうかを迅速に検出することができる。タグ復号論理804によるソフトデュアルタグの検出は、「E」ビットまたはソフトデュアル操作符号を選択してフォーマット論理806から命令#2アライメントおよびリマッピング論理818に通すようにマルチプレクサ808を制御する。シングルアドレッシング論理810およびデュアルアドレッシング論理812は並列に作動して、命令の左端から常に所定のオフセットで配置されるアドレスフィールドの復号を開始することができる。デュアル復号論理802およびソフトデュアルタグフィールド復号論理804の出力は、論理814によって結合され、マルチプレクサ816への制御入力を形成する。したがって、デュアル命令が検出されると、デュアルアドレッシング論理812の出力はDAGENコントロールに通され、そうでなければ、シングルアドレッシング論理810の出力がDAGENコントロールに通される。

【0104】上述したように、別の形式では、複合命令は、この複合命令を形成する一対の命令の別々のDAGEN符号を置換する結合DAGEN符号を含むことができる。複合命令内のDAGENタグは結合DAGEN符号フィールドの存在を識別することができ、デコーダはDAGENタグに応答して結合DAGEN符号フィールドを復号するように構成されている。結合DAGEN符号フィールドは結合アドレスフィールドの一部を形成することができる。結合DAGENフィールドにより、実行速度が有利になる。

【0105】命令がソフトデュアル命令であるならば、復号を実施する前にリマッピングが必要である。したがって、命令フィールドリマッピング論理824は、ソフトデュアルタグ復号論理804の出力に応答して、その対の第1の命令に関連する情報のリマッピングを行った後に、そのリマップされた操作情報を第1の命令用の復号論理826に通す。同様に、命令対の第2の命令のための命令アライメントおよびリマッピング論理818が、ソフトデュアルタグ復号論理804の出力に応答して、第2のメモリ命令に関連する情報のリマッピングを行った後に、その情報を第2の命令用の復号論理822に通す。命令アライメントおよびフィールドリマッピング論理818は、適切なビット16，ビット24，ビット32またはビット40の命令境界に従って第1の命令のフォーマットに応じて第2の命令をリアラインするように作動することもできる。

【0106】図10および図13を参照すると、図13に示す復号機構は命令バッファからの命令を復号するように構成されている。図10に示すように、復号機構は、ソフトデュアル命令のタグフィールド内の所定のタグに応答して、所定のソフトデュアル命令内の複合アドレスフィールドからの第1のメモリ命令に対する第1のメモリアドレスおよび第2のメモリ命令に対する第2のメモリアドレスを復号する。

【0107】並列イネーブルビット復号論理820は、第2の命令を第1の命令と並列に復号して実行できるかどうかを検証するように作動する。ソフトデュアル命令は並列イネーブル（「E」）ビットを含まないため、ソフトデュアル命令が検出されると、この論理820はディセーブルされる。

【0108】図14は、ソフトデュアル命令にインターフェイスするメモリバスの態様を示す略ブロック図であり、図15はソフトデュアル命令用のオペランドフェッチ制御を要約する表である。

【0109】図14は、Cバス750，Dバス752，Eバス760およびFバス762を示し、これらのバスは、前に参照されているが、個別に識別されてはいない。

【0110】ソフトデュアルフェッチコントローラ754は、プロセッサコア102の命令制御機能の一部を形成する。それは、オペランドフェッチ機構756，782を制御して、第1のデータフローパス790に対するXおよびYオペランド758，780と第2のデータフローパス792に対するXおよびYオペランド784，786とをCおよびDバス750，752を介してそれぞれフェッチするように作動する。やはりプロセッサコア102の命令制御機能の一部を形成するソフトデュアルライトコントローラ755は、メモリライトインターフェイス794，796を制御して、第1のデータフローパス790および第2のデータフローパス792からEバス760およびFバス762へのオペランドの各書込みを制御する。

【0111】図15の表は、ソフトデュアルフェッチコントローラ754によって行われるオペランドフェッチ制御操作を示す。これは、スタンドアロンで行われる単一メモリ命令と比べた場合のソフトデュアルメモリ命令に対するオペランドフェッチフローへの変化を示す。したがって、単一メモリ命令がスタンドアロンで実行される場合には、オペランドレジスタはDバスからロードされ、それにより、メモリ要求はDバスとなるので、2サイクルを要する。しかしながら、ソフトデュアル命令が実行されると、フェッチコントローラは、Ymemバスに対するオペランドフェッチフローを変えて、要求がC要求へ再度向けられ、1500に示すようにオペランドがDバスではなくCバスからフェッチされるようにする。オペランド#1およびオペランド#2は同じサイク

ル内で並列にフェッチされる。同じ機構がライトインターフェイスに適用される。たとえば、Eバス要求をFバス要求に再指向することができる。

【0112】図16は、図1のプロセッサ10を内蔵する集積回路40の略図である。集積回路は、特定用途集積回路（ASIC）技術を使用して実現することができる。図から分かるように、集積回路は表面実装用の複数のコンタクト42を含んでいる。しかしながら、集積回路は他の構成を含むことができ、たとえばゼロ挿入力ソケット内に搭載するための回路下面上の複数のピン、または他の任意適切な構成とすることができる。

【0113】たとえば図16の集積回路に内蔵されているプロセッサ10のような処理エンジンの1つの応用は、たとえば移動体ワイヤレス電気通信装置のような電気通信装置である。図17にこのような電気通信装置の一例を示す。図17に示す特定の例では、電気通信装置は、キーパッドまたはキーボード12およびディスプレイ14のような一体型ユーザ入力装置を有する移動体電話機11である。ディスプレイは、たとえば液晶ディスプレイやTFTディスプレイのような適切な技術を使用して実現することができる。プロセッサ10はキーパッド12に接続され、そこで、適切なキーボードアダプタ（不図示）を介してディスプレイ14に接続され、そこで、適切なディスプレイアダプタ（不図示）を介して電気通信インターフェイスまたはトランシーバ16、たとえば無線周波数（RF）回路を含むワイヤレス電気通信インターフェイスに接続されている。無線周波数回路は、プロセッサ10を含む集積回路40に内蔵しても、そこから分離してもよい。RF回路16はアンテナ18に接続されている。

【0114】ソフト符号化デュアルメモリアクセス命令を実行する処理エンジンについて説明してきた。ソフトデュアル命令機構により2つのメモリアクセス命令を高い符号化効率で並列に実行することができる。並列性が増すため、消費電力を低減することができる。また、第2の命令に対するデコーダは第1の命令に対するデコーダのサブセットとすることができ、シリコンリアルエステートが効率的に使用され、消費電力をさらに低減することができる。

【0115】ここで使用した「印加される」、「接続される」および「接続」という用語は、電気的接続パス内に付加要素がある場合も含めて、電気的に接続されることを意味する。

【0116】実施例について本発明を説明してきたが、本明細書に制約的な意味合いはない。当業者ならば、本明細書を読めば本発明の他のさまざまな実施例が自明であろう。したがって、本発明の真の範囲および精神に含まれる実施例のこのようないかなる修正も添付した特許請求の範囲に含まれるものとする。

【0117】本出願は欧州で1998年10月6日に出願されたS．N．98402456．2（TI－27685EU）および欧州で1998年10月6日に出願されたS．N．98402455．4（TI－28433EU）に優先権を請求するものである。

【図面の簡単な説明】

【図1】本発明の実施例に従ったプロセッサの略ブロック図である。

【図2】図1のプロセッサのコアの略図である。

【図3】図1のプロセッサのコアのさまざまな実行ユニットのより詳細な略ブロック図である。

【図4】図1のプロセッサの命令バッファキューおよび命令デコーダコントローラの略図である。

【図5】図1のプロセッサのパイプラインフェーズの表現である。

【図6】図1のプロセッサにおけるパイプラインの動作例の線図である。

【図7】図1のプロセッサのパイプラインの動作を説明するためのプロセッサのコアの略表現である。

【図8】命令対の例を示す図である。

【図9】さまざまな命令に対するバスサイクルの相対タイミングを示す図である。

【図10】ソフトデュアル命令の実行例を示す図である。

【図11】ソフトデュアル命令の発生を示す略図である。

【図12】ソフトデュアル命令発生のフロー図である。

【図13】ソフトデュアル命令を実行する構造のブロック図である。

【図14】ソフトデュアル命令操作をインターフェイスするメモリバスを示す図である。

【図15】ソフトデュアル命令のオペランドフェッチ制御を示す表である。

【図16】図1のプロセッサを内蔵する集積回路の略図である。

【図17】図1のプロセッサを内蔵する電気通信装置の略図である。

【符号の説明】

10　マイクロプロセッサ
20　プロセッサバックプレーン
22　バックプレーンバス
24　命令キャッシュメモリ
26　周辺装置
28　外部インターフェイス
30　レジスタファイル
32　データアドレス発生サブユニット
34　ALU
36　Dユニットレジスタファイル
38　DユニットALU
40　Dユニットシフタ
42，44　累算ユニット

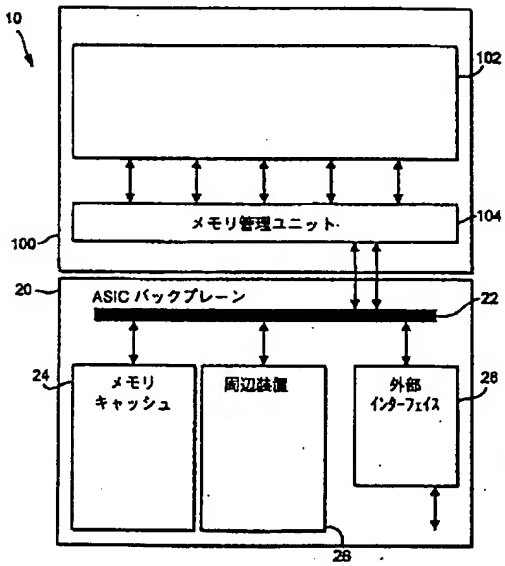
25

100 処理エンジン
102 処理コア
104 インターフェイスユニット
106 命令バッファユニット
108 プログラムフローユニット
110 アドレスデータフローユニット
112 データ通信ユニット
118 アドレスバス
120 データバス
122 プログラムリードバス
128 プログラムアドレスバス
130，132 データライトバス
140 命令定数データバス
146，148 累算器ライトギス
150，152 累算器リードバス
160，162 データライトアドレスバス
502 命令バッファキュー
504 レジスタ
512，514 命令デコーダ
520，521 マルチプレクサ
530 ライトプログラムカウンタ
532 ローカルライトプログラムカウンタ

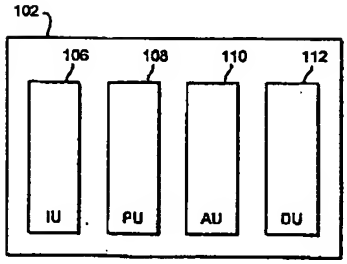
26

534 リードプログラムカウンタ
536 ローカルリードプログラムカウンタ
750 Cバス
752 Dバス
754 ソフトデュアルフェッチコントローラ
755 ソフトデュアルライトコントローラ
756，782 オペランドフェッチ機構
760 Eバス
762 Fバス
790，792 データフローパス
794，796 メモリライトインターフェイス
802 デュアル復号論理
804 ソフトデュアルタグフィールド復号論理
806 フォーマット論理
808，816 マルチプレクサ
810 シングルアドレッシング論理
812 デュアルアドレッシング論理
814 論理
818 命令アライメントおよびフィールドリマッピング論理
822，826 復号論理
824 命令フィールドリマッピング論理

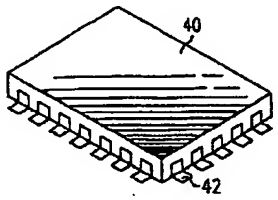
【図1】



【図2】



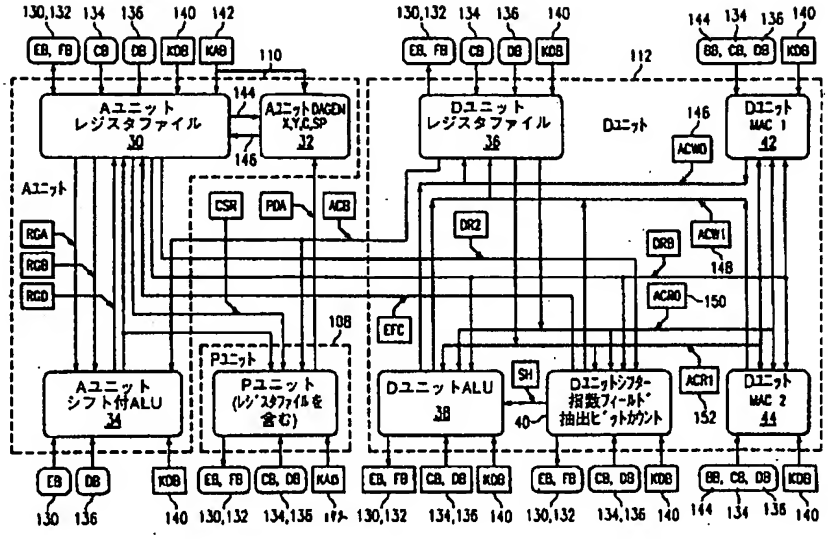
【図16】



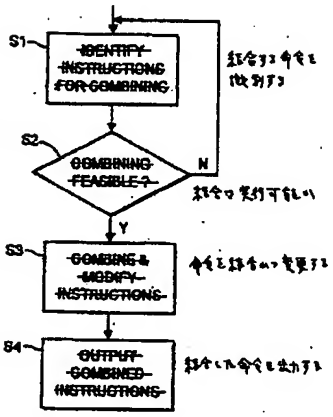
【図8】

命令 Instruction #1	命令 Instruction #2	例 Example	並列性 Parallelism
Single-Memory シングルメモリ	Register レジスタ	ACy = rnd (ACy + (Smem *ACx)) || dst = dst + src	// enable イネーブル
Single-Memory シングルメモリ	Control 制御	ACy = rnd (ACy + (Smem *ACx)) || repeat (k8)	// enable イネーブル
Single-Memory シングルメモリ	Stack スタック	ACy = rnd (ACy + (Smem *ACx)) || push (src)	// enable イネーブル
Single-Memory シングルメモリ	Single-Memory シングルメモリ	ACx = rnd (Xmem *Xmem) || ACy = ACx + (Smem << DRx)	Soft-Dual ソフトデュアル
Dual-Memory デュアルメモリ	Register レジスタ	ACx = (Xmem <<#16) + (Ymem <<#16) || DRy = DRx & k8	// enable イネーブル
Dual-Memory デュアルメモリ	Control 制御	dbl (Ymem) - dbl (Xmem) || goto L8	// enable イネーブル
Dual-Memory デュアルメモリ	Stack スタック	ACy = rnd (ACy + Smem *ACx)) || push (src1 src2)	// enable イネーブル

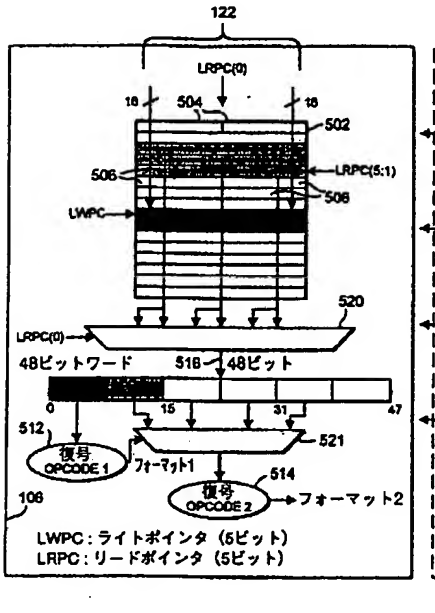
【図3】



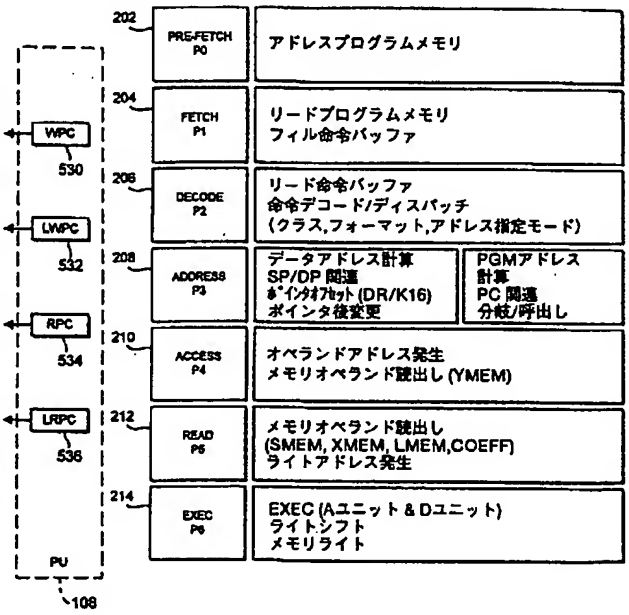
【図12】



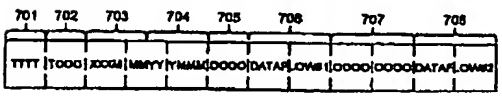
【図4】



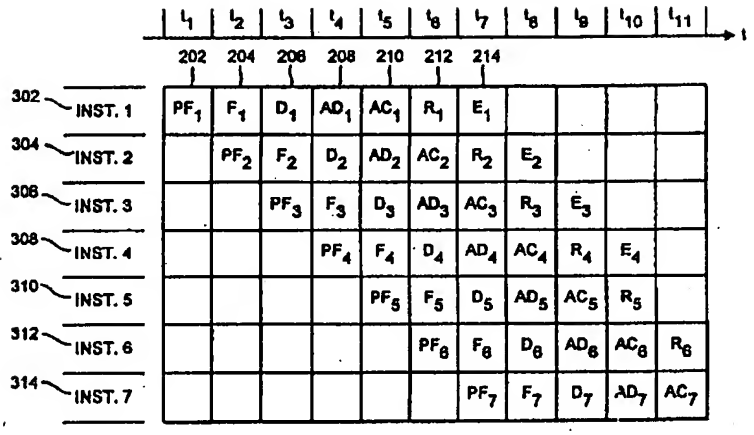
【図5】



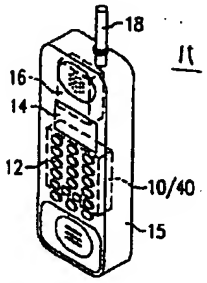
【図10】



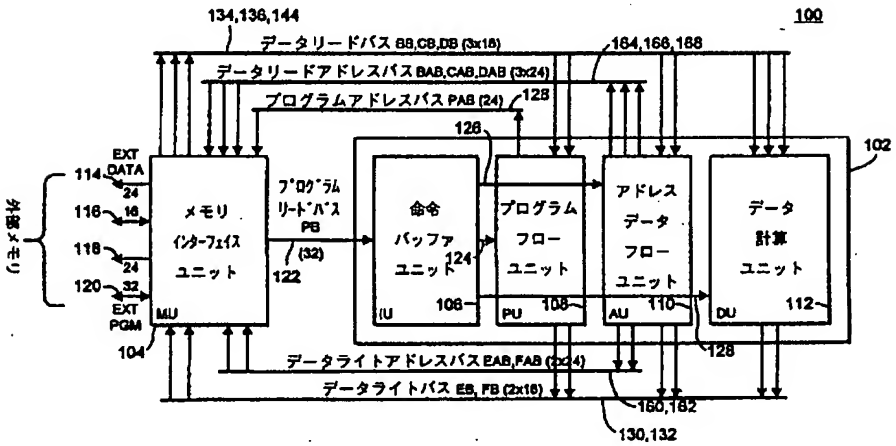
【図6】


t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t
202 204 206 208 210 212 214
302 INST. 1 PF1 F1 D1 AD1 AC1 R1 E1
304 INST. 2 PF2 F2 D2 AD2 AC2 R2 E2
306 INST. 3 PF3 F3 D3 AD3 AC3 R3 E3
308 INST. 4 PF4 F4 D4 AD4 AC4 R4 E4
310 INST. 5 PF5 F5 D5 AD5 AC5 R5
312 INST. 6 PF6 F6 D6 AD6 AC6 R6
314 INST. 7 PF7 F7 D7 AD7 AC7

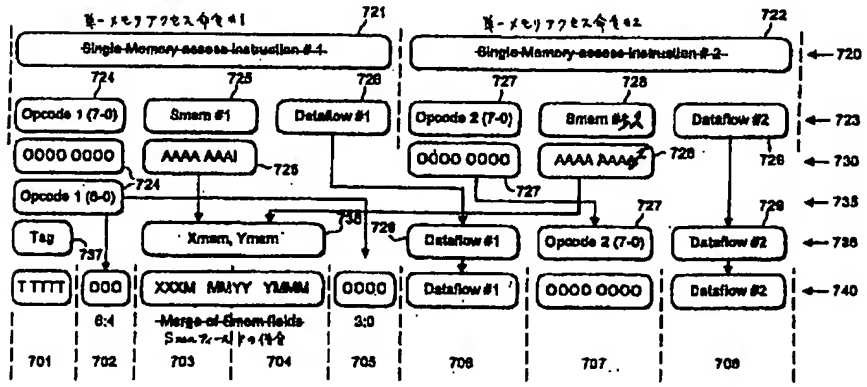
【図17】


18
11
16
14
12
10/40
15

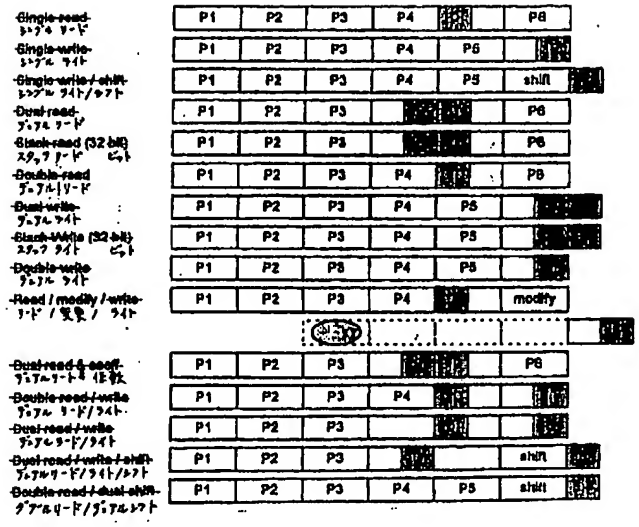
【図7】


134,136,144
100
データリードバス BB,CB,DB (3x16)
164,166,168
データリードアドレスバス BAB,CAB,DAB (3x24)
プログラムアドレスバス PAB (24)
128
126
102
EXT DATA
114
24
116
16
118
24
120
32
EXT PGM
104
メモリ インターフェイス ユニット
MU
プログラム リードバス PB
(32)
122
命令 バッファ ユニット
IU
124
106
プログラム フロー ユニット
PU
108
アドレス データ フロー ユニット
AU
110
データ 計算 ユニット
DU
112
データライトアドレスバス EAB,FAB (2x24)
データライトバス ES, FB (2x16)
160,162
130,132

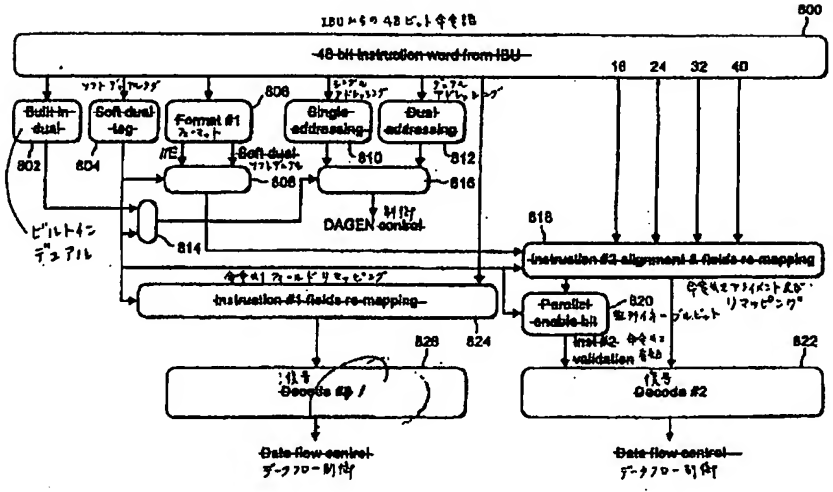
【図11】


721
Single-Memory-access-instruction-#-1
722
Single-Memory-access-instruction-#-2
720
724
725
726
727
728
Opcode 1 (7-0)
Smem #1
Dataflow #1
Opcode 2 (7-0)
Smem #2
Dataflow #2
723
0000 0000
AAAA AAAI
AAAA AAA0
730
Opcode 1 (5-0)
735
Tag
737
Xmem, Ymem
738
740
TTTTT
000
XXXM MMYY YMMM
0000
6:4
3:0
Merge-of-Smem-fields
701
702
703
704
705
706
707
708

【図9】



【図13】

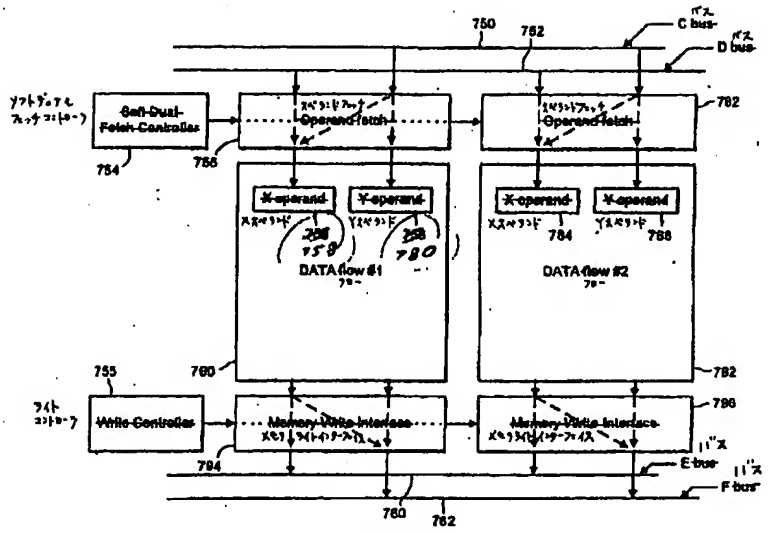


【図15】

DAGEN #1	DAGEN #2	ソフトデュアル soft-dual-DAGEN	Operand #1 standalone fetch	Operand #2 standalone fetch	Operand #1 soft-dual-fetch	Operand #2 soft-dual-fetch
Smem_R	Smem_R	Dual_RR	DB		DB	
Smem_R	Smem_W	Dual_RW	DB	-	DB	-
Smem_R	Smem_WF	Dual_RWF	DB	-	DB	-
Lmem_R	Lmem_W	Dual_RW	CB, DB	-	CB, DB	-
Lmem_R	Lmem_WF	Dual_RWF	CB, DB	-	CB, DB	-

1500

【図14】



フロントページの続き

(72)発明者　カリム　ドジャファリアン
フランス国，バチマン　ビー1，レ　トス　カヌ，ブールバール　ド　ラ　レイヌ　ジャンヌ，453

(72)発明者　エルブ　カタン
フランス国，サン　ローラン　デュ　バル，コルニシュ　ファネストク，1050

(72)発明者　バンサン　ジレ
フランス国，ル　ルレ，シュマン　デ　ムタン，6

【外国語明細書】

Compound Memory Access Instructions

This application claims priority to S.N. 98402456.2, filed in Europe on October 6, 1998 (TI-27685EU) and S.N. 98402455.4, filed in Europe on October 6, 1998 (TI-28433EU).

Field of the Invention

The present invention relates to processing engines, and to the parallel execution of instructions in such processing engines.

Background of the Invention

It is known to provide for parallel execution of instructions in microprocessors using multiple instruction execution units. Many different architectures are known to provide for such parallel execution. Providing parallel execution increases the overall processing speed. Typically, multiple instructions are provided in parallel in an instruction buffer and these are then decoded in parallel and are dispatched to the execution units. Microprocessors are general purpose processing engines which require high instruction throughputs in order to execute software running thereon, which can have a wide range of processing requirements depending on the particular software applications involved. Moreover, in order to support parallelism, complex operating systems have been necessary to control the scheduling of the instructions for parallel execution.

Many different types of processing engines are known, of which microprocessors are but one example. For example, Digital Signal Processors (DSPs) are widely used, in particular for specific applications. DSPs are typically configured to optimize the

performance of the applications concerned and to achieve this they employ more specialized execution units and instruction sets.

The present invention is directed to improving the performance of processing engines such as for example, but not exclusively, digital signal processors.

Summary of the Invention

Particular and preferred aspects of the invention are set out in the accompanying independent and dependent claims. Combinations of features from the dependent claims may be combined with features of the independent claims as appropriate and not merely as explicitly set out in the claims.

In accordance with a first aspect of the invention, there is provided a processing engine comprising an instruction buffer operable to buffer single and compound instructions pending execution thereof, and a decode mechanism configured to decode instructions from the instruction buffer. The decode mechanism is configured to be responsive to a predetermined tag in a tag field of an instruction, which predetermined tag is representative of the instruction being a compound instruction formed from separate programmed memory instructions. The decode mechanism is operable in response to the predetermined tag to decode at least a first data flow control for a first programmed instruction and a second data flow control for a second programmed instruction.

Thus, an embodiment of the invention provides a decode mechanism responsive to compound instructions formed (e.g., assembled or compiled) by combining separate programmed instructions. In this manner, it is possible to optimize the use of the bandwidth available within the processing engine. Appropriate programmed instructions, such as suitable memory instructions, can thus be assembled, or compiled, to form a compound instruction. By generating a separate control flow for each of the constituent programmed instructions from the compound instruction, those instructions can be performed wholly or partially in parallel with a positive effect on the overall throughput of the processing engine. The control flow generated by the decode mechanism for each of the programmed instructions can be the same as that which would have been generated for the programmed instructions if they had been held as single instructions in the instruction buffer.

A compact and efficient encoding can be enabled in an embodiment of the invention. For example by ensuring that a memory instruction can only be a first of a pair of instructions in the instruction buffer in the form of a predetermined compound

instruction, parallelism of memory access instructions can be provided with efficient encoding, efficient use of real estate and reduced power consumption.

In an embodiment of the invention, the compound instruction is defined as a soft compound memory instruction formed by combining (e.g. using an instruction preprocessing mechanism such as a compiler or an assembler) from separate programmed memory instructions. In a particular example, the compound instruction is a soft dual memory instruction, that is a dual memory instruction assembled from separate first and second programmed memory instructions, although in other examples more than two instructions can be assembled into a compound instruction.

Preferably, the decode mechanism is operable to decode a first memory address for a first programmed memory address instruction and a second memory address for a second programmed memory instruction from a compound memory address field in the compound instruction. Particularly, where the compound address field of the compound instruction is at the same bit positions as the address field for a hard programmed dual memory instruction, this can have a positive effect on instruction throughput. In this case the decoding of the addresses can be started before the operation code of the instructions have been decoded regardless of the format of first and second instructions of a dual instruction.

In order to reduce the number of bits required for the compound instruction, the memory addresses in the compound address field of the compound instruction can be arranged to be indirect addresses, whereby the decode mechanism needs only to be operable to decode indirect addresses for such instructions. As dual instructions support less options than single instructions, the size of a post modification field for the addresses can be reduced, thereby reducing the number of bits required for the addresses themselves and also to dispense with an indirect/direct indicator bit.

A memory access instruction can be constrained to be a first instruction of a pair of instructions in the instruction buffer. In this case a soft dual instruction effectively provides an encoding corresponding to two memory instructions. As a result, the need for a parallel enable field can be avoided, any memory instruction being implicitly capable of parallelism. This also provides further advantages of providing a reduction of an

application code size, with optimization of external interface bandwidth and a reduction of cache misses.

The decoder for the second instruction of an instruction pair can also be made as a subset of the decoder for the first instruction resulting in a reduction in the integrated circuit real estate required and a reduction in power consumption for the processing engine.

In order to provide a compact instruction format and to enable the address field to be located at the same position as for a hard compound instruction, the compound instruction can comprise a split operation code field for a first instruction of the predetermined compound instruction. The operation code can be spilt either side of the address field, for example. The decoder can be response to detection of the appropriate tag field to decode the split operation code for the first instruction of the compound instruction.

In order to further reduce the number of bits, the compound instruction can comprise a reduced operation code field for at least the first instruction of the predetermined compound instruction such that the operation code field comprises fewer bits that the operation code field of the first programmed instruction. By restricting the range of operation codes for memory instructions to be within a certain range or ranges, the number of bits which need to be provided for the first operation code can be reduced. The decode mechanism can be arranged to be responsive to the predetermined tag to decode a reduced size operation code for the first instruction of the compound instruction.

With the various measures mentioned above, the predetermined compound instruction can be arranged to have the same number of bits in total as the sum of the bits of the separate programmed instructions. Reorganization of the fields from the programmed instructions can lead to the predetermined compound instruction having a common overall format with other instructions.

Where each programmed instruction has a data address generation (DAGEN) code field, the individual DAGEN codes of the individual programmed instructions could be combined into a combined DAGEN code field within the compound instruction. This could provide more rapid decoding and execution of the compound instruction. The

combined DAGEN code field could form part of a combined address field. Where a combined DAGEN code field is provided, the decode mechanism can be operable to respond to a predetermined DAGEN tag to decode the combined DAGEN field.

The processing engine can be provided with a data fetch controller operable to fetch, in parallel, first and second operands from addresses identified by the first and second memory addresses, respectively. A data write controller can also be operable to write in parallel the result of first and second data flow operations for the first and second instructions, respectively. Also, dual read/write operations can be provided.

In an embodiment of the invention, assembler syntax can differentiate between hard compound and soft compound syntax to provide visibility for available slots for parallelism. A hard compound instruction can be executed in parallel with a non-memory instruction such as a control flow or register instruction as indicated by a parallel enable bit and as long as there are no bus/operator resource conflicts.

In accordance with another aspect of the invention, there is provided a processor, for example, but not necessarily, a digital signal processor, comprising a processing engine as described above. The processor can be implemented as an integrated circuit, for example as an Application Specific Integrated Circuit (ASIC).

A digital signal processing system comprising a processing engine as described above can also be provided with an instruction preprocessing mechanism operable to combine separate programmed memory instructions to form a compound memory instruction. The instruction preprocessor can be in the form of a compiler, assembler, etc., which is operable to compile or assemble compound instructions from programmed instructions. The mechanism can be configured to be operable to determine whether the separate programmed memory instructions may be combined prior to assembly of the compound instruction.

In accordance with a further aspect of the invention, there is provided an instruction preprocessor for a digital signal processing system, the instruction preprocessor being configured to be operable:

to determine programmed memory instructions capable of being combined; and

to assemble a compound memory instruction from said determined programmed memory instructions.

It should be understood that in the present context the term "instruction preprocessor" is to be understood broadly to cover any mechanism for preprocessing instructions, that is compiling and/or assembling instructions, including compilers, assemblers, etc.

The instruction preprocessor may be provided separately, for example on a carrier medium such as a data storage medium (a disc, solid state memory, a data transmission medium such as an electrical, optical or other electromagnetic (e.g. wireless transmission medium)).

In accordance with another aspect of the invention, there is provided a method of improving the performance of a processing engine. The method includes:

buffering a compound instruction assembled from separate programmed memory instructions, the compound instruction including a tag field containing a predetermined compound instruction tag; and

responding to the predetermined compound instruction tag in the tag field of an instruction in the instruction buffer to decode, from the compound instruction, at least first data flow control for a first programmed instruction and second data flow control for a second programmed instruction.

Brief Description of the Drawings

Particular embodiments in accordance with the invention will now be described, by way of example only, and with reference to the accompanying drawings in which like reference signs are used to denote like parts, unless otherwise stated, and in which:

Figure 1 is a schematic block diagram of a processor in accordance with an embodiment of the invention;

Figure 2 is a schematic diagram of a core of the processor of Figure 1;

Figure 3 is a more detailed schematic block diagram of various execution units of the core of the processor of Figure 1;

Figure 4 is a schematic diagram of an instruction buffer queue and an instruction decoder controller of the processor of Figure 1;

Figure 5 is a representation of pipeline phases of the processor of Figure 1;

Figure 6 is a diagrammatic illustration of an example of the operation of a pipeline in the processor of Figure 1;

Figure 7 is a schematic representation of the core of the processor for explaining the operation of the pipeline of the processor of Figure 1;

Figure 8 illustrates examples of instruction pairs;

Figure 9 illustrates the relative timing of bus cycles for various instructions;

Figure 10 illustrates an example of the execution of a soft dual instruction;

Figure 11 is a schematic diagram illustrating the generation of a soft dual instruction.

Figure 12 is a flow diagram of the generation of a soft dual instruction;

Figure 13 is a block diagram of a structure for executing a soft dual instruction;

Figure 14 illustrates memory bus interfacing for a soft dual instruction operation;

Figure 15 is a table illustrating operand fetch control for a soft dual instruction.

Figure 16 is a schematic representation of an integrated circuit incorporating the processor of Figure 1; and

Figure 17 is a schematic representation of a telecommunications device incorporating the processor of Figure 1.

Description of Particular Embodiments

Although the invention finds particular application to Digital Signal Processors (DSPs), implemented for example in an Application Specific Integrated Circuit (ASIC), it also finds application to other forms of processing engines.

Figure 1 is a block diagram of a microprocessor 10 which has an embodiment of the present invention. Microprocessor 10 is a digital signal processor ("DSP"). In the interest of clarity, Figure 1 only shows those portions of microprocessor 10 that are relevant to an understanding of an embodiment of the present invention. Details of general construction for DSPs are well known, and may be found readily elsewhere. For example, U.S. Patent 5,072,418 issued to Frederick Boutaud, et al, describes a DSP in detail and is incorporated herein by reference. U.S. Patent 5,329,471 issued to Gary Swoboda, et al, describes in detail how to test and emulate a DSP and is incorporated herein by reference. Details of portions of microprocessor 10 relevant to an embodiment of the present invention are explained in sufficient detail hereinbelow, so as to enable one of ordinary skill in the microprocessor art to make and use the invention.

Several example systems which can benefit from aspects of the present invention are described in U.S. Patent 5,072,418, which was incorporated by reference herein, particularly with reference to Figures 2-18 of U.S. Patent 5,072,418. A microprocessor incorporating an aspect of the present invention to improve performance or reduce cost can be used to further improve the systems described in U.S. Patent 5,072,418. Such systems include, but are not limited to, industrial process controls, automotive vehicle systems, motor controls, robotic control systems, satellite telecommunication systems, echo canceling systems, modems, video imaging systems, speech recognition systems, vocoder-modem systems with encryption, and such.

A description of various architectural features and a description of a complete set of instructions of the microprocessor of Figure 1 is provided in co-assigned application Serial No. 98402455.9 (TI-28433), which is incorporated herein by reference.

The basic architecture of an example of a processor according to the invention will now be described.

Figure 1 is a schematic overview of a processor 10 forming an exemplary embodiment of the present invention. The processor 10 includes a processing engine 100 and a processor backplane 20. In the present embodiment, the processor is a Digital Signal Processor 10 implemented in an Application Specific Integrated Circuit (ASIC).

As shown in Figure 1, the processing engine 100 forms a central processing unit (CPU) with a processing core 102 and a memory interface, or management, unit 104 for interfacing the processing core 102 with memory units external to the processor core 102.

The processor backplane 20 comprises a backplane bus 22, to which the memory management unit 104 of the processing engine is connected. Also connected to the backplane bus 22 is an instruction cache memory 24, peripheral devices 26 and an external interface 28.

It will be appreciated that in other embodiments, the invention could be implemented using different configurations and/or different technologies. For example, the processing engine 100 could form the processor 10, with the processor backplane 20 being separate therefrom. The processing engine 100 could, for example be a DSP separate from and mounted on a backplane 20 supporting a backplane bus 22, peripheral and external interfaces. The processing engine 100 could, for example, be a microprocessor rather than a DSP and could be implemented in technologies other than ASIC technology. The processing engine, or a processor including the processing engine, could be implemented in one or more integrated circuits.

Figure 2 illustrates the basic structure of an embodiment of the processing core 102. As illustrated, the processing core 102 includes four elements, namely an Instruction Buffer Unit (I Unit) 106 and three execution units. The execution units are a Program Flow Unit (P Unit) 108, Address Data Flow Unit (A Unit) 110 and a Data Computation Unit (D Unit) 112 for executing instructions decoded from the Instruction Buffer Unit (I Unit) 106 and for controlling and monitoring program flow.

Figure 3 illustrates the P Unit 108, A Unit 110 and D Unit 112 of the processing core 102 in more detail and shows the bus structure connecting the various elements of the processing core 102. The P Unit 108 includes, for example, loop control circuitry, GoTo/Branch control circuitry and various registers for controlling and monitoring

program flow such as repeat counter registers and interrupt mask, flag or vector registers. The P Unit 108 is coupled to general purpose Data Write busses (EB, FB) 130, 132, Data Read busses (CB, DB) 134, 136 and an address constant bus (KAB) 142. Additionally, the P Unit 108 is coupled to sub-units within the A Unit 110 and D Unit 112 via various busses labeled CSR, ACB and RGD.

As illustrated in Figure 3, in the present embodiment the A Unit 110 includes a register file 30, a data address generation sub-unit (DAGEN) 32 and an Arithmetic and Logic Unit (ALU) 34. The A Unit register file 30 includes various registers, among which are 16 bit pointer registers (AR0-AR7) and data registers (DR0-DR3) which may also be used for data flow as well as address generation. Additionally, the register file includes 16 bit circular buffer registers and 7 bit data page registers. As well as the general purpose busses (EB, FB, CB, DB) 130, 132, 134, 136, a data constant bus 140 and address constant bus 142 are coupled to the A Unit register file 30. The A Unit register file 30 is coupled to the A Unit DAGEN unit 32 by unidirectional busses 144 and 146 respectively operating in opposite directions. The DAGEN unit 32 includes 16 bit X/Y registers and coefficient and stack pointer registers, for example for controlling and monitoring address generation within the processing engine 100.

The A Unit 110 also comprises the ALU 34 which includes a shifter function as well as the functions typically associated with an ALU such as addition, subtraction, and AND, OR and XOR logical operators. The ALU 34 is also coupled to the general-purpose busses (EB, DB) 130, 136 and an instruction constant data bus (KDB) 140. The A Unit ALU is coupled to the P Unit 108 by a PDA bus for receiving register content from the P Unit 108 register file. The ALU 34 is also coupled to the A Unit register file 30 by busses RGA and RGB for receiving address and data register contents and by a bus RGD for forwarding address and data registers in the register file 30.

As illustrated, the D Unit 112 includes a D Unit register file 36, a D Unit ALU 38, a D Unit shifter 40 and two multiply and accumulate units (MAC1, MAC2) 42 and 44. The D Unit register file 36, D Unit ALU 38 and D Unit shifter 40 are coupled to busses (EB, FB, CB, DB and KDB) 130, 132, 134, 136 and 140, and the MAC units 42 and 44 are coupled to the busses (CB, DB, KDB) 134, 136, 140 and data read bus (BB) 144.

The D Unit register file 36 includes 40-bit accumulators (AC0-AC3) and a 16-bit transition register. The D Unit 112 can also utilize the 16 bit pointer and data registers in the A Unit 110 as source or destination registers in addition to the 40-bit accumulators. The D Unit register file 36 receives data from the D Unit ALU 38 and MACs 1&2 42, 44 over accumulator write busses (ACW0, ACW1) 146, 148, and from the D Unit shifter 40 over accumulator write bus (ACW1) 148. Data is read from the D Unit register file accumulators to the D Unit ALU 38, D Unit shifter 40 and MACs 1&2 42, 44 over accumulator read busses (ACR0, ACR1) 150, 152. The D Unit ALU 38 and D Unit shifter 40 are also coupled to sub-units of the A Unit 108 via various busses labeled EFC, DRB, DR2 and ACB.

Referring now to Figure 4, there is illustrated an instruction buffer unit 106 comprising a 32 word instruction buffer queue (IBQ) 502. The IBQ 502 comprises 32x16 bit registers 504, logically divided into 8 bit bytes 506. Instructions arrive at the IBQ 502 via the 32-bit program bus (PB) 122. The instructions are fetched in a 32-bit cycle into the location pointed to by the Local Write Program Counter (LWPC) 532. The LWPC 532 is contained in a register located in the P Unit 108. The P Unit 108 also includes the Local Read Program Counter (LRPC) 536 register, and the Write Program Counter (WPC) 530 and Read Program Counter (RPC) 534 registers. LRPC 536 points to the location in the IBQ 502 of the next instruction or instructions to be loaded into the instruction decoder(s) 512 and 514. That is to say, the LRPC 534 points to the location in the IBQ 502 of the instruction currently being dispatched to the decoders 512, 514. The WPC points to the address in program memory of the start of the next 4 bytes of instruction code for the pipeline. For each fetch into the IBQ, the next 4 bytes from the program memory are fetched regardless of instruction boundaries. The RPC 534 points to the address in program memory of the instruction currently being dispatched to the decoder(s) 512 and 514.

The instructions are formed into a 48-bit word and are loaded into the instruction decoders 512, 514 over a 48-bit bus 516 via multiplexors 520 and 521. It will be apparent to a person of ordinary skill in the art that the instructions may be formed into words

comprising other than 48-bits, and that the present invention is not limited to the specific embodiment described above.

The bus 516 can load a maximum of two instructions, one per decoder, during any one instruction cycle. The combination of instructions may be in any combination of formats, 8, 16, 24, 32, 40 and 48 bits, which will fit across the 48-bit bus. Decoder 1, 512, is loaded in preference to decoder 2, 514, if only one instruction can be loaded during a cycle. The respective instructions are then forwarded on to the respective function units in order to execute them and to access the data for which the instruction or operation is to be performed. Prior to being passed to the instruction decoders, the instructions are aligned on byte boundaries. The alignment is done based on the format derived for the previous instruction during decoding thereof. The multiplexing associated with the alignment of instructions with byte boundaries is performed in multiplexors 520 and 521.

The processor core 102 executes instructions through a 7 stage pipeline, the respective stages of which will now be described with reference to Figure 5.

The first stage of the pipeline is a PRE-FETCH (P0) stage 202, during which stage a next program memory location is addressed by asserting an address on the address bus (PAB) 118 of a memory interface, or memory management unit 104.

In the next stage, FETCH (P1) stage 204, the program memory is read and the I Unit 106 is filled via the PB bus 122 from the memory management unit 104.

The PRE-FETCH and FETCH stages are separate from the rest of the pipeline stages in that the pipeline can be interrupted during the PRE-FETCH and FETCH stages to break the sequential program flow and point to other instructions in the program memory, for example for a Branch instruction.

The next instruction in the instruction buffer is then dispatched to the decoder/s 512/514 in the third stage, DECODE (P2) 206, where the instruction is decoded and dispatched to the execution unit for executing that instruction, for example to the P Unit 108, the A Unit 110 or the D Unit 112. The decode stage 206 includes decoding at least part of an instruction including a first part indicating the class of the instruction, a second part indicating the format of the instruction and a third part indicating an addressing mode for the instruction.

The next stage is an ADDRESS (P3) stage 208, in which the address of the data to be used in the instruction is computed, or a new program address is computed should the instruction require a program branch or jump. Respective computations take place in the A Unit 110 or the P Unit 108 respectively.

In an ACCESS (P4) stage 210 the address of a read operand is output and the memory operand, the address of which has been generated in a DAGEN X operator with an Xmem indirect addressing mode, is then READ from indirectly addressed X memory (Xmem).

The next stage of the pipeline is the READ (P5) stage 212 in which a memory operand, the address of which has been generated in a DAGEN Y operator with an Ymem indirect addressing mode or in a DAGEN C operator with coefficient address mode, is READ. The address of the memory location to which the result of the instruction is to be written is output.

In the case of dual access, read operands can also be generated in the Y path, and write operands in the X path.

Finally, there is an execution EXEC (P6) stage 214 in which the instruction is executed in either the A Unit 110 or the D Unit 112. The result is then stored in a data register or accumulator, or written to memory for Read/Modify/Write or store instructions. Additionally, shift operations are performed on data in accumulators during the EXEC stage.

The basic principle of operation for a pipeline processor will now be described with reference to Figure 6. As can be seen from Figure 6, for a first instruction 302, the successive pipeline stages take place over time periods T_1-T_7. Each time period is a clock cycle for the processor machine clock. A second instruction 304, can enter the pipeline in period T_2, since the previous instruction has now moved on to the next pipeline stage. For instruction 3, 306, the PRE-FETCH stage 202 occurs in time period T_3. As can be seen from Figure 6 for a seven stage pipeline a total of 7 instructions may be processed simultaneously. For all 7 instructions 302-314, Figure 6 shows them all under process in time period T_7. Such a structure adds a form of parallelism to the processing of instructions.

As shown in Figure 7, the present embodiment of the invention includes a memory management unit 104 which is coupled to external memory units (not shown) via a 24 bit address bus 114 and a bi-directional 16 bit data bus 116. Additionally, the memory management unit 104 is coupled to program storage memory (not shown) via a 24 bit address bus 118 and a 32 bit bi-directional data bus 120. The memory management unit 104 is also coupled to the I Unit 106 of the machine processor core 102 via a 32 bit program read bus (PB) 122. The P Unit 108, A Unit 110 and D Unit 112 are coupled to the memory management unit 104 via data read and data write busses and corresponding address busses. The P Unit 108 is further coupled to a program address bus 128.

More particularly, the P Unit 108 is coupled to the memory management unit 104 by a 24 bit program address bus 128, the two 16 bit data write busses (EB, FB) 130, 132, and the two 16 bit data read busses (CB, DB) 134, 136. The A Unit 110 is coupled to the memory management unit 104 via two 24 bit data write address busses (EAB, FAB) 160, 162, the two 16 bit data write busses (EB, FB) 130, 132, the three data read address busses (BAB, CAB, DAB) 164, 166, 168 and the two 16 bit data read busses (CB, DB) 134, 136. The D Unit 112 is coupled to the memory management unit 104 via the two data write busses (EB, FB) 130, 132 and three data read busses (BB, CB, DB) 144, 134, 136.

Figure 7 represents the passing of instructions from the I Unit 106 to the P Unit 108 at 124, for forwarding branch instructions for example. Additionally, Figure 7 represents the passing of data from the I Unit 106 to the A Unit 110 and the D Unit 112 at 126 and 128 respectively.

In a particular embodiment of the invention, the processing engine 100 is responsive to machine instructions in a number of formats. Examples of such instructions in different formats are illustrated in the following.

8 Bit instruction: OOOO OOOO

This represents an eight bit instruction, for example a memory map qualifier (MMAP()) or a read port qualifier (readport()). Such a qualifier comprises merely an eight bit opcode (OOOO OOOO). In such a case parallelism is implicit.

16 Bit Instruction: OOOO OOOE FSSS FDDD

This represents an example of a sixteen bit instruction, for example an instruction where the content of a destination register (e.g., dst) becomes the sum of the prior content of that register (dst) and the content of a source register (src), that is:

dst = dst + src

Such an instruction comprises a seven bit opcode (OOOO OOO) with a one bit parallel enable field (E), a four bit source register identifier (FSSS) and a four bit destination register identifier (FDDD).

16 Bit Instruction: OOOO FDDD PPPM MMMI

This represents another example of a sixteen bit instruction, for example where the content of a destination register (e.g., dst) becomes the content of a memory location (Smem), that is:

dst = Smem

Such an instruction comprises a four bit opcode (OOOO), a four bit destination register identifier (FDDD), a three bit pointer address (PPP), a four bit address modifier (MMMM) and a direct/indirect address indicator (I).

24 Bit Instruction: OOOO OOOE LLLL LLLL oCCC CCCC

This represents an example of a twenty four bit instruction, for example a conditional instruction for a branch to and offset (L8) where a condition is met, that is:

if(cond) goto L8

Such an instruction comprises a seven bit opcode (OOOO OOO) with a one bit parallel enable field (E), an eight bit branch offset (LLLL LLLL), a one bit opcode extension (o) and a seven bit condition field (CCC CCCC).

24 Bit Instruction: OOOO OOOO PPPM MMMI SSDD ooU%

This is another example of a twenty-four bit instruction, for example a single memory operand instruction where the content of an accumulator (AC_y) becomes the

result of rounding the sum of the content of another accumulator (AC_x) and the square of the content of a memory location (with optional rounding), and optionally the content of a data register (DR3) can become the content of the memory location, that is:

$$AC_y = rnd(AC_x * Smem * Smem), DR3 = Smem$$

Such an instruction comprises an eight bit opcode (OOOO OOOO), a three bit pointer address (PPP), a four bit address modifier (MMMM), a one bit direct/indirect address indicator field (I), a two bit source accumulator identifier (SS), a two bit destination accumulator identifier (DD), a two bit opcode extension (oo), an update condition field (u), and a one bit rounding option field (%).

32 Bit Instruction: OOOO OOOO PPPM MMMI KKKK KKKK KKKK KKKK

This is an example of a thirty-two bit instruction, for example an instruction where the content of a test register (TC1) is set to 1 or 0 depending on the sign comparison of a memory location (Smem) to a constant value (K16), that is:

$$TC1 = (Smem == K16)$$

Such an instruction comprises an eight bit opcode (OOOO OOOO), a three bit pointer address (PPP), a four bit address modifier (MMMM), a one bit direct/indirect address indicator field (I) and a sixteen bit constant field (KKKK KKKK KKKK KKKK).

Hard Dual Instruction: OOOO OOOO XXXM MMYY YMMM SSDD ooox ssU%

This is an example of a 32 bit dual access instruction, which could be termed a "hard dual access instruction", or a hard programmed dual memory instruction, that is a dual instruction which has been programmed as such, for example, by a programmer. Such an instruction requires two DAGEN operators. A second instruction can be executed in parallel. This is typically a register or control instruction. Memory stack instructions can also be executed in parallel as long as there are no bus conflicts. An example of such an instruction is:

$$C_y = rnd(DR_x * Xmem),$$
$$Ymem = HI(AC_x << DR2)$$

DR3 = Xmem

This instruction comprises an eight bit opcode (OOOO OOOO), a three bit Xmem pointer address (XXX) with a four bit address modifier (MMMM), a three bit Ymem pointer address (YYY) with a four bit address modifier (MMMM), a two bit source accumulator (AC_x) identifier (SS), a two bit destination accumulator (AC_y) identifier (DD), a three bit opcode extension (ooo), a don't care bit (x), a two bit source accumulator identifier (ss), a one bit optional DR3 update field (U) and a one bit optional rounding field (%).

Figure 8 is a table illustrating combinations of instructions forming instruction pairs and also a soft dual instruction. In such instruction pairs, the first instruction of the pair is always a memory operation. It will be noted that where the second instruction is also a memory instruction, then this is configured as a soft dual instruction, that is a compound instruction.

Instructions which may be located in a second position of an instruction pair (i.e. for the higher program address of the pair) include a parallel enable field (E bit) to indicate whether the instruction can be performed in parallel with the first of a pair of instructions. The parallel enable bit is located at a predetermined offset from the instruction format boundary between the instructions. The decoder is arranged to be responsive to the 'E' bit in order to control instruction execution.

The reason for having a memory operation first in an instruction pair is that at the entry to the address decode stage of the processor pipeline, the decoder does not know the format of the instruction, or even where the format boundary is located. Memory address decoding is one of the critical stages of the pipeline to ensure good instruction throughput. Accordingly, it is necessary to be able reliably to know the location and size of the address bits for a memory instruction to be decoded in order that the decoding can commence even before the exact nature of the instruction is determined.

A further advantage which results from constraining a memory instruction to be located as the first instruction in an instruction pair is that it is then not necessary for a memory instruction to include a field indicating whether parallel operation is permitted. This makes the instruction set more efficient and allows improved code size.

Yet a further advantage is that the hardware necessary for decoding a second instruction of an instruction pair need only be a subset of the hardware for decoding the first instruction of the instruction pair. The first instruction is the instruction of the instruction pair with a lower program address than the second instruction of the instruction pair. Thus, the decode hardware for the instruction with a higher program address of an instruction pair can be a subset of the decode hardware for the instruction with a lower program address of an instruction pair. This enables a reduction in the silicon area and power consumption required for implementing and operating the decode hardware.

Where two instructions of an instruction pair can be performed in parallel, this takes place in respective decoding and execution stages. However, due to physical bus timing constraints, bus transfers can be staggered.

Figure 9 illustrates the pipeline stage in which memory access takes place for different types of instructions, including dual instructions. It should be noted, as for Figure 4, that the pipeline stages shown are for illustrative purposes only. In practice, the prefetch and fetch stages form a flow separate from that of the remaining stages.

Comparing Figure 9 with Figure 5, P1 represents the fetch stage, P2 the decode stage, P3 the address computation stage, P4 the access stage, P5 the read stage and P6 the execute stage. B represents a coefficient read access from a register via the B bus. C and D represent memory read accesses via the C and D busses respectively. E and F represent write accesses via the E and F busses respectively. In order that the read and write accesses can be performed at the required cycles without causing a bubble (or stall) on the pipeline, decoding is performed as early as possible.

Figure 10 illustrates a particular form of dual memory access instruction. It is effectively formed from two merged programmed instructions which have implied parallelism. The dual memory instruction of Figure 10 is termed a soft dual instruction, or also a compound instruction herein. It is formed by combining two programmed single memory access instructions in an instruction preprocessor, for example in a compiler or an assembler. In other words, this compound instruction is not programmed, or pre-programmed, as a dual instruction by a programmer. This provision of this form of

compound instruction enables improved memory access performance by permitting parallel operation, with both instructions being executed in the same cycle. In a particular example described in the following, the soft dual instruction is restricted to indirect addressing with dual modifier options. As a result, it is possible to encode the soft dual instruction to achieve increased performance through parallel operation with no size penalty in respect of the combined instruction size.

The soft dual instruction is qualified by a five bit tag field 701, with individual following instruction fields organized as illustrated in Figure 10. The size of the tag field results from constraints relating to the particular implementation, namely:

- that the total encoding format is constrained not be greater than the sum of the encoding formats of the two constituent programmed instructions;
- that the total instruction format size is a multiple of 8; and
- the availability of opcodes with respect to other single instructions.

Following the tag field 701 are:

- part 702 of the operation code field for a first instruction;
- a compound address field 703/704 including an indirect memory address (XXXMMM) 703 for the first instruction and an indirect memory address (YYYMMM) 704 for a second instruction;
- the remainder of the operation code field 705 for the first instruction;
- a data flow field 706 for the first instruction;
- an operation code field 707 for the operation code of the second instruction; and
- a data flow field 708 for the second instruction.

It can be seen, therefore, that the combined address portion for the soft dual instruction is held at the same location in the soft dual instruction as for any other dual instruction. This provides the advantage of rapid address decoding as a result of being able to commence address decoding without knowledge of the instruction type involved. It will be seen that in order to achieve this, some reorganization of the bits in the soft dual instruction is necessary, for example as described above.

In addition to the modifications described above, where two programmed instructions each comprise a data address generation (DAGEN) field, these could be

combined to form a combined DAGEN field in the soft dual instruction. The provision of a combined DAGEN field can facilitate and speed subsequent execution of the soft dual instruction.

Figure 11 illustrates various steps in transforming two independent instructions into a soft dual instruction.

Two independent instructions 721 and 722 are represented at stage 720.

As shown at 723, a first 24 bit instruction 721 includes an eight bit operation code 724 in the first byte, a single memory (Smem) address 725 in the next byte and data flow bits 726 in the next byte. A second 24 bit instruction 722 includes an eight bit operation code 727 in the first byte, a single memory address 728 in the next byte and data flow bits 729 in the next byte. At 730, the eight operation code bits are each labeled 'O' in the operation code bytes 724 and 727 of each of the instructions. The single memory addresses 725 and 728 are each shown to comprise 7 address bits 'A' plus an indirect/direct indicator bit 'I'. This is because addresses for the standard memory accesses can be either direct or indirect. In the example shown, the granularity is based on bytes. However, in other examples a granularity based on other than 8 bits may be employed.

At stage 735, the operation code 724 of the first instruction is split into two parts. Only seven of the eight bits of the operation code 724 need to be considered. This is as a result of memory code mapping which can ensure that this is redundant in the case of a soft dual instruction (e.g., by ensuring that all memory instructions have operation codes within a determined range, for example, 80-FF in hexadecimal notation, for a soft dual instruction). As can be seen later in stages 736 and 740, and also in Figure 10 the operation code for the first instruction is split. Three bits of the operation code for the first instruction are placed between a soft dual instruction tag 737 and the combined addresses 738 for the first and second instructions and four bits are placed after the combined addresses 738.

At stage 736, the insertion of a soft dual instruction tag 737 is shown. This as a tag which can be interpreted by the decoder as representing a soft dual instruction. Also shown is the merging of the single memory fields 725 and 728. This can be achieved

because all soft dual instructions are restricted to indirect addresses, whereby an indirect/direct flag is not needed. The indirect addresses are indicated by a three bit base address XXX or YYY, for the first and second instructions, respectively, and a three bit modifier (MMM). Stage 736 further illustrates the moving of the data flow for the first instruction to the first byte position of the second instruction, with the operation code for the second instruction being moved to the second byte position of that instruction.

As a result, the format of the soft dual instruction represented in Figure 10 is achieved. It is to be noted that there is no code size penalty for a soft dual instruction versus two single memory access instructions. By replacing two single memory (Smem) instructions by an Xmem, Ymem, enough bits are freed up to insert the 'soft dual' tag 701/737. The soft dual tag by itself allows the decoder to detect that it should decode the pair of instructions as memory instructions. Instruction set mapping can be used to ensure that memory instructions are encoded within a window 80-FF, whereby the most significant bit (bit 7) of the first operation code 724 can be discarded when effecting the dual field encoding.

In the example shown, the various stages illustrated in Figure 11 are performed by an instruction preprocessor, for example a compiler or an assembler, when preparing instructions for execution. The steps performed by the instruction preprocessor are represented in a flow diagram shown in Figure 12.

In step S1, the instruction preprocessor detects the presence of two instructions which might potentially be combined into a soft dual instruction. In order for this to be possible, the instructions will need to be such that they may be performed in parallel and do not result in data or control flow anomalies. Each instruction within the instruction set is qualified by DAGEN variables in a DAGEN tag, which define the address generator resources and the type of memory access involved to support the instruction.

Accordingly, in step S2, the instruction preprocessor performs a first step in determining the feasibility of merging two standalone memory instructions into a soft dual instruction by analyzing the DAGEN variables. Assuming this checks out, then the instruction preprocessor is operable to analyze potential bus and operator conflicts and to

establish whether there is a potential bar to the combining of the first and second instructions.

In step S3, the instruction preprocessor then applies the soft dual instruction tag 737 and modifies the operation codes and address indications, as well as the field positions as illustrated in Figure 11.

In step S4, the soft dual instruction is output by the instruction preprocessor.

Figure 13 is a schematic block diagram illustrating the decoding process for a soft dual instruction. Figure 13 illustrates the decoding of a 48 bit instruction word 800 from the instruction buffer unit 106.

From the operation code (opcode), which is located at the left of the instruction word as shown in Figure 13, logic 802, 804 in the opcode decoding circuitry is able rapidly to detect whether a built in dual or soft dual instruction is to be decoded. The detection of a soft dual tag by tag decoding logic 804 controls a multiplexor 808 to select either an "E" bit or the soft dual opcodes to be passed from format logic 806 to instruction #2 alignment and remapping logic 818. Single addressing logic 810 and dual addressing logic 812 are operable in parallel to commence decoding of the address fields, which are always located at a determined offset from the left hand end of the instruction. Outputs of dual decoding logic 802 and soft dual tag field decoding logic 804 are combined by logic 814 and form a control input to a multiplexor 816. Thus, when a dual instruction is detected, the output of dual addressing logic 812 is passed to the DAGEN control, otherwise the output of single addressing logic 810 is passed to DAGEN control.

As mentioned above, in an alternative form, a compound instruction can comprise a combined DAGEN code field replacing the separate DAGEN codes of the pair of instructions forming the compound instruction. A DAGEN tag in the compound instruction could identify the presence of the combined DAGEN code field, with the decoder being configured to be responsive to the DAGEN tag to decode the combined DAGEN code field. The combined DAGEN code field could form part of the combined address field. The provision of a combined DAGEN field can provide advantages in execution speed.

If the instruction is a soft dual instruction, then remapping is necessary before decoding can be performed. Accordingly, instruction field remapping logic 824 is responsive to the output of the soft dual tag decoding logic 804 to cause the remapping of the information relating to the first instruction of the pair before passing the remapped operation information to decode logic 826 for the first instruction. Similarly, instruction alignment and remapping logic 818 for a second instruction of the instruction pair is responsive to the output of the soft dual tag decoding logic 804 to cause remapping of the information relating to the second memory instruction prior to passing the information to the decode logic 822 for the second instruction. The instruction alignment and field remapping logic 818 is also operable to realign the second instruction dependent upon the format of the first instruction, according to the instruction boundary at bit 16, bit 24, bit 32 or bit 40, as appropriate.

With reference to Figures 10 and 13, it can be seen that the decode mechanism shown in Figure 13 is configured to decode instructions from the instruction buffer. The decode mechanism is responsive to a predetermined tag in a tag field of a soft dual instruction as shown in Figure 10 to decode a first memory addresses for a first memory instruction and a second memory address for a second memory instruction from a compound address field in the predetermined soft dual instruction.

Parallel enable bit decoding logic 820 is operable to validate whether the second instruction may be decoded and executed in parallel with the first instruction. As a soft dual instruction does not include a parallel enable ("E") bit, this logic 820 is disabled when a soft dual instruction is detected.

Figure 14 is a schematic block diagram illustrating aspects of the memory bus interfacing for a soft dual instruction, and Figure 15 is a table summarizing the operand fetch control for a soft dual instruction.

Figure 14 illustrates the C bus 750, the D bus 752, the E bus 760 and the F bus 762, which busses were referenced earlier, but were not individually identified.

A soft dual fetch controller 754 forms part of the instruction control functions of the processor core 102. This is operable to control operand fetch mechanisms 756 and 782 to fetch X and Y operands 758 and 780 for a first data flow path 790, and X and Y

operands 784 and 786 for a second data flow path 792, respectively, via the C and D busses 750 and 752. A soft dual write controller 755, which also forms part of the instruction control functions of the processor core 102, is operable to control memory write interfaces 794 and 796 to control the writing of operands from the first data flow path 790 and the second data flow path 792, respectively to the E and F busses 760 and 762.

The table which forms Figure 15 illustrates the operand fetch control operations performed by the soft dual fetch controller 754. This illustrates the changes to the operand fetch flow for a soft dual memory instruction compared to a single memory instruction performed standalone. Thus, when a single memory instruction is executed standalone, the operand register is loaded from the D bus, whereby the memory request is a D request, thereby requiring two cycles. However, when a soft dual instruction is executed, the fetch controller changes the operand fetch flow for the Ymem path, such that the request is re-directed to a C request and the operand is fetched from the C bus instead of the D bus as indicated at 1500. Advantageously, operand #1 and operand #2 are fetched in parallel in the same cycle. The same mechanism applies to the write interface. For example, an E bus request can be redirected to an F bus request.

Figure 16 is a schematic representation of an integrated circuit 40 incorporating the processor 10 of Figure 1. The integrated circuit can be implemented using application specific integrated circuit (ASIC) technology. As shown, the integrated circuit includes a plurality of contacts 42 for surface mounting. However, the integrated circuit could include other configurations, for example a plurality of pins on a lower surface of the circuit for mounting in a zero insertion force socket, or indeed any other suitable configuration.

One application for a processing engine such as the processor 10, for example as incorporated in an integrated circuit as in Figure 16, is in a telecommunications device, for example a mobile wireless telecommunica-tions device. Figure 17 illustrates one example of such a telecommunications device. In the specific example illustrated in Figure 17, the telecommunica-tions device is a mobile telephone 11 with integrated user input device such as a keypad, or keyboard 12 and a display 14. The display could be implemented

using appropriate technology, as, for example, a liquid crystal display or a TFT display. The processor 10 is connected to the keypad 12, where appropriate via a keyboard adapter (not shown), to the display 14, where appropriate via a display adapter (not shown), and to a telecommunications interface or transceiver 16, for example a wireless telecommunications interface including radio frequency (RF) circuitry. The radio frequency circuitry could be incorporated into, or separate from, an integrated circuit 40 comprising the processor 10. The RF circuitry 16 is connected to an aerial 18.

Thus, there has been described a processing engine which provides for execution of soft encoded dual memory access instructions. The soft dual instruction mechanism enables execution of two memory access instructions in parallel with high encoding efficiency. Due to increased parallelism, power consumption can be reduced. Also, a decoder for a second instruction can be a subset of the decoder for a first instruction resulting in efficient use of silicon real estate and providing further opportunities for a reduction in power consumption.

As used herein, the terms "applied," "connected," and "connection" mean electrically connected, including where additional elements may be in the electrical connection path.

While the invention has been described with reference to illustrative embodiments, this description is not intended to be construed in a limiting sense. Various other embodiments of the invention will be apparent to persons skilled in the art upon reference to this description. It is therefore contemplated that the appended claims will cover any such modifications of the embodiments as fall within the true scope and spirit of the invention.

What is Claimed is:

1. A digital system comprising a processing engine, wherein the processing engine comprises:

an instruction buffer operable to buffer single and compound instructions pending execution thereof; and

a decode mechanism configured to decode instructions from the instruction buffer;

the decode mechanism being responsive to a predetermined tag in an instruction, the predetermined tag being representative of the instruction being a compound instruction formed from separate programmed memory instructions, to decode at least first data flow control for a first programmed instruction and at least second data flow control for a second programmed instruction.

2. The processing engine according to claim 1, wherein the compound instruction is a compound memory instruction formed by combining separate first and second programmed memory instructions.

3. The processing engine according to claim 2, wherein the decode mechanism is operable to decode a first memory address for a first programmed memory address instruction and a second memory address for a second programmed memory instruction from a compound memory address field in the compound instruction.

4. The processing engine according to claim 3, wherein the compound address field of the compound instruction is at the same bit positions as the address field for a hard programmed dual memory instruction.

5. The processing engine according to claim 4, wherein the memory addresses in the compound address field of the compound instruction are indirect addresses, the decode mechanism being operable to decode the indirect addresses.

6. The processing engine according to claim 1, wherein the compound instruction comprises a split operation code field for a first programmed instruction of the compound instruction.

7. The processing engine according to claim 6, wherein the decode mechanism is responsive to the predetermined tag to decode a split operation code for the first programmed instruction of the compound instruction.

8. The processing engine according to claim 7, wherein the compound instruction comprises an operation code field for a first programmed instruction of the compound instruction, which operation code field comprises less bits than the operation code field of the first programmed instruction.

9. The processing engine according to claim 8, wherein the decode mechanism is responsive to the predetermined tag to decode a reduced size operation code for the first programmed instruction of the compound instruction.

10. The processing engine according to claim 9, wherein the compound instruction has the same number of bits in total as the sum of the bits of the separate programmed instructions.

11. The processing engine according to claim 1, wherein the compound instruction has a combined data address generation (DAGEN) field formed from DAGEN fields of the first and second programmed memory instructions.

12. The processing engine according to claim 11, wherein the combined DAGEN field forms part of a combined address field.

13. The processing engine according to claim 12, wherein the decode mechanism is responsive to a predetermined DAGEN tag to decode the combined DAGEN field.

14. The processing engine according to claim 1, comprising a fetch controller operable to fetch in parallel first and second operands from addresses identified by the first and second memory addresses, respectively.

15. The processing engine according to claim 14, comprising a write controller operable to write in parallel the result of first and second data flow operations for the first and second programmed instructions, respectively.

16. The processing engine according to claim 1, operable to interpret a memory access instruction as implicitly capable of parallel execution, whereby a memory access instruction does not including a parallel enable field.

17. The processing engine according to claim 1, wherein a memory access instruction is constrained to be a first programmed instruction of a pair of instructions in the instruction buffer.

18. The digital system of Claim 1 being a cellular telephone, further comprising:

an integrated keyboard connected to the processor via a keyboard adapter;

a display, connected to the processor via a display adapter;

radio frequency (RF) circuitry connected to the processor; and

an aerial connected to the RF circuitry.

19. The digital system of Claim 1, further comprising an instruction preprocessing means for preparing instructions for execution, the instruction

preprocessing means being operable to combine separate programmed memory instructions to form a compound memory instruction.

20. A method of improving the performance of a processing engine, the method comprising the steps of:

buffering a compound instruction formed from separate programmed memory instructions, the compound instruction including a tag field containing a predetermined compound instruction tag; and

responding to the predetermined compound instruction tag in the tag field of an instruction in the instruction buffer to decode, from the compound instruction, at least first data flow control for a first programmed instruction and second data flow control for a second programmed instruction.

21. The method according to claim 20, further comprising the step of combining separate first and second programmed memory instructions to form the compound instruction.

22. The method according to claim 20, further comprising the step of decoding at least a first memory address for the first programmed memory instruction and a second memory address for the second programmed memory instruction from a compound address field of the compound instruction.

23. The method according to claim 22, further comprising the step of decoding the compound address field of the compound instruction from the same bit positions as for the address field for a hard programmed dual memory instruction.

24. The method according to claim 20, further comprising the step of decoding a split operation code for a first instruction of the compound instruction.

25. The method according to claim, further comprising decoding a reduced size operation code for the first instruction of the compound instruction.

26. The method according to claim 21, wherein the step of responding comprises decoding a combined data address generation (DAGEN) field formed from DAGEN fields of the first and second programmed memory instructions.

27. The method according to claim 26, wherein the combined DAGEN field forms part of a combined address field.

28. The method according to claim 26, wherein the decode mechanism is responsive to a predetermined DAGEN tag to decode the combined DAGEN field.

29. The method according to claim 22, further comprising the step of fetching in parallel first and second operands from addresses identified by first and second memory addresses, respectively.

30. The method according to claim 29, comprising writing in parallel the result of first and second data flow operations for first and second programmed instructions, respectively, of the compound instruction.

31. The method according to claim 21, wherein the step of combining comprises determining whether the separate programmed memory instructions may be combined prior to assembly of the compound instruction.

32. The method according to claim 31, wherein the step of combining further comprises:

determining programmed memory instructions capable of being combined; and

combining the determined programmed memory instructions to form a compound memory instruction.

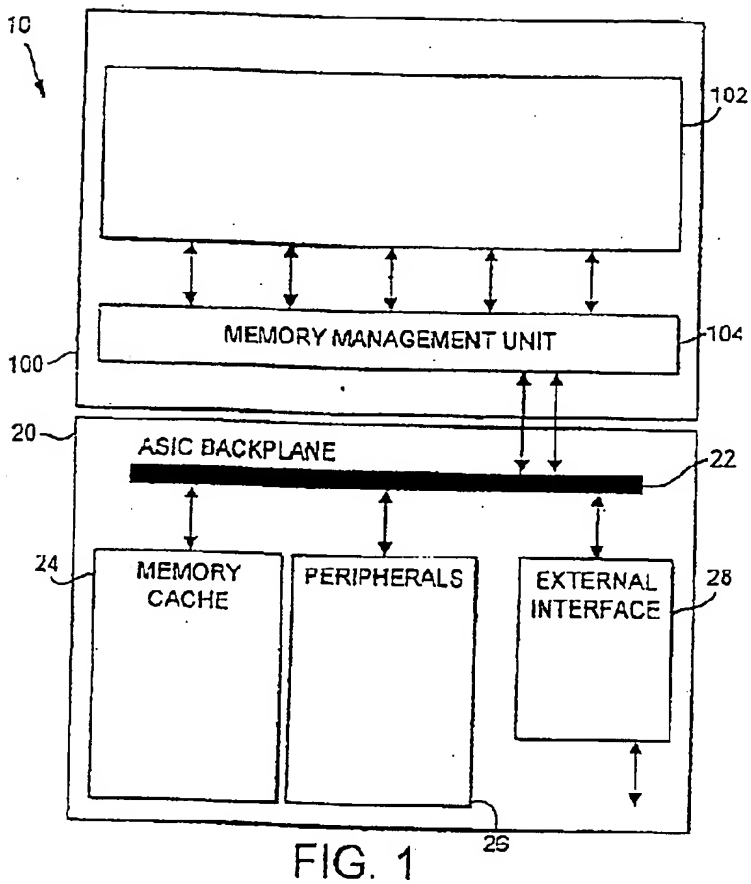

10
102
MEMORY MANAGEMENT UNIT
104
100
20
ASIC BACKPLANE
22
24
MEMORY CACHE
PERIPHERALS
EXTERNAL INTERFACE
28
26

FIG. 1

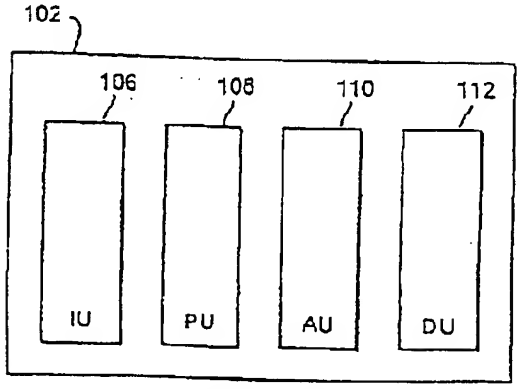

102
106
108
110
112
IU
PU
AU
DU

FIG. 2

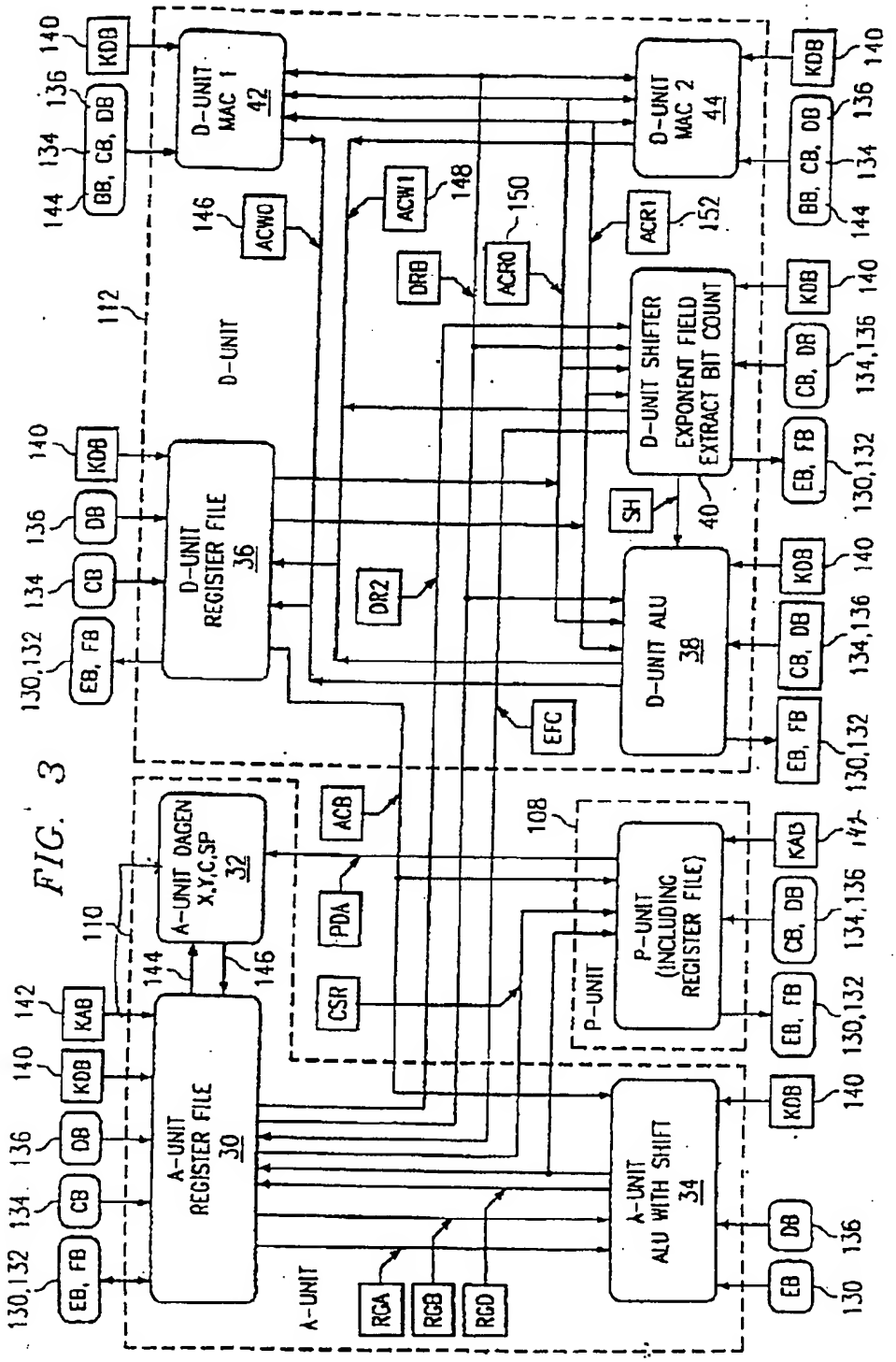

D-UNIT
D-UNIT MAC 1 42
D-UNIT MAC 2 44
D-UNIT REGISTER FILE 36
D-UNIT SHIFTER EXPONENT FIELD EXTRACT BIT COUNT
D-UNIT ALU 38
A-UNIT DAGEN X,Y,C,SP 32
A-UNIT REGISTER FILE 30
A-UNIT ALU WITH SHIFT 34
P-UNIT
P-UNIT (INCLUDING REGISTER FILE)
KDB
BB, CB, DB
DB
CB
EB, FB
KAB
KAB
ACWO
ACW1
ACRO
ACR1
DRB
DR2
SH
EFC
ACB
PDA
CSR
RGA
RGB
RGD
CB, DB
CB, DR
EB
130,132
134,136
140
144
146
148
150
152
108
110
112

FIG. 3

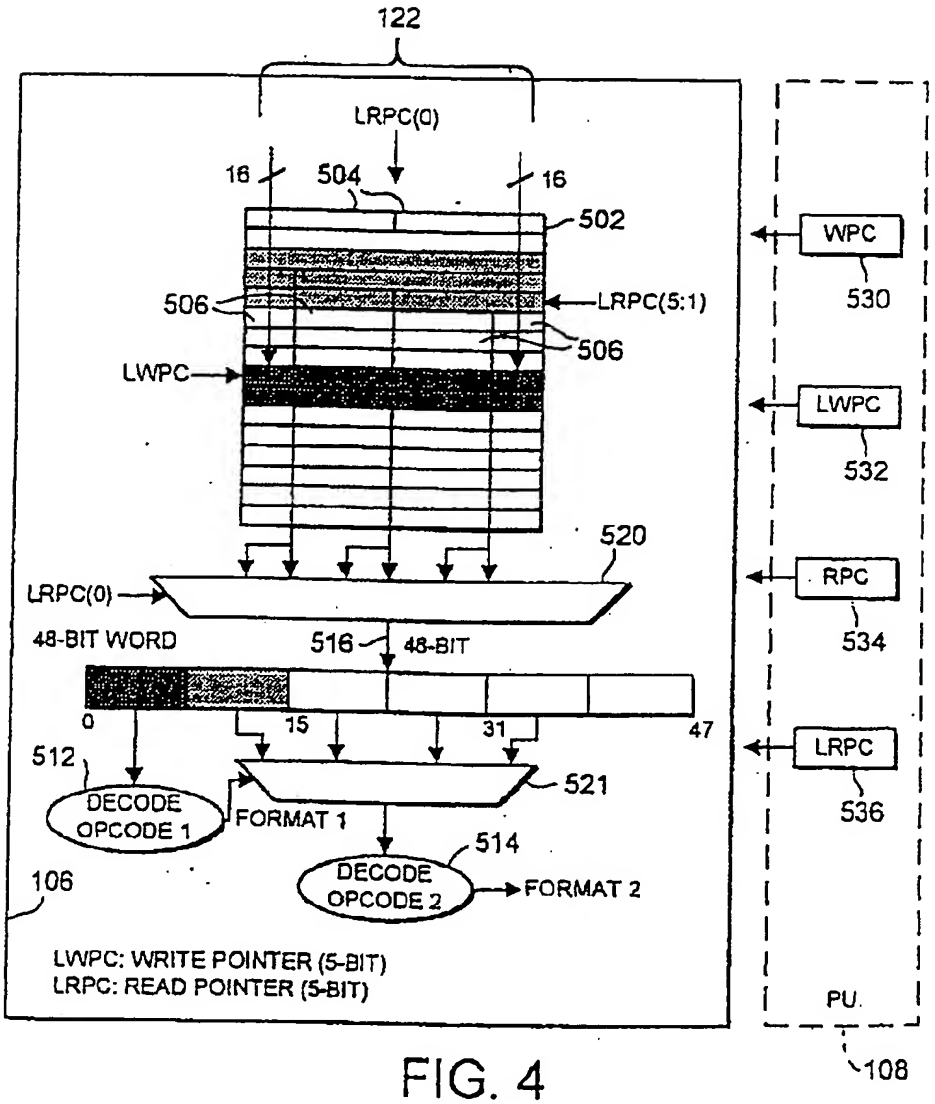

122
LRPC(0)
16
504
16
502
506
LRPC(5:1)
506
LWPC
520
LRPC(0)
48-BIT WORD
516
48-BIT
0
15
31
47
512
DECODE OPCODE 1
FORMAT 1
521
514
DECODE OPCODE 2
FORMAT 2
106
LWPC: WRITE POINTER (5-BIT)
LRPC: READ POINTER (5-BIT)
WPC
530
LWPC
532
RPC
534
LRPC
536
PU
108

FIG. 4

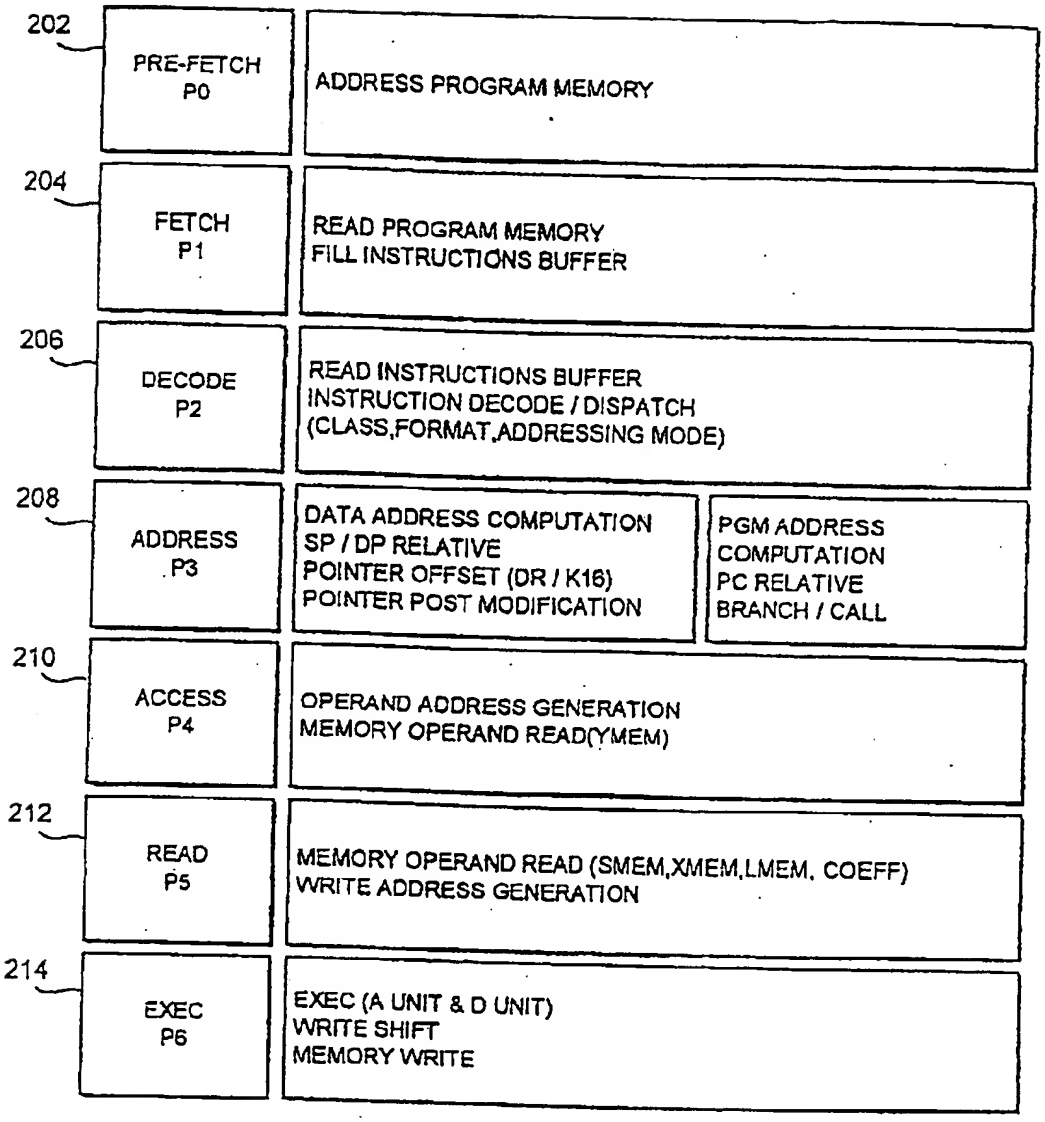

202
PRE-FETCH
P0
ADDRESS PROGRAM MEMORY
204
FETCH
P1
READ PROGRAM MEMORY
FILL INSTRUCTIONS BUFFER
206
DECODE
P2
READ INSTRUCTIONS BUFFER
INSTRUCTION DECODE / DISPATCH
(CLASS,FORMAT,ADDRESSING MODE)
208
ADDRESS
P3
DATA ADDRESS COMPUTATION
SP / DP RELATIVE
POINTER OFFSET (DR / K16)
POINTER POST MODIFICATION
PGM ADDRESS
COMPUTATION
PC RELATIVE
BRANCH / CALL
210
ACCESS
P4
OPERAND ADDRESS GENERATION
MEMORY OPERAND READ(YMEM)
212
READ
P5
MEMORY OPERAND READ (SMEM,XMEM,LMEM, COEFF)
WRITE ADDRESS GENERATION
214
EXEC
P6
EXEC (A UNIT & D UNIT)
WRITE SHIFT
MEMORY WRITE

FIG. 5

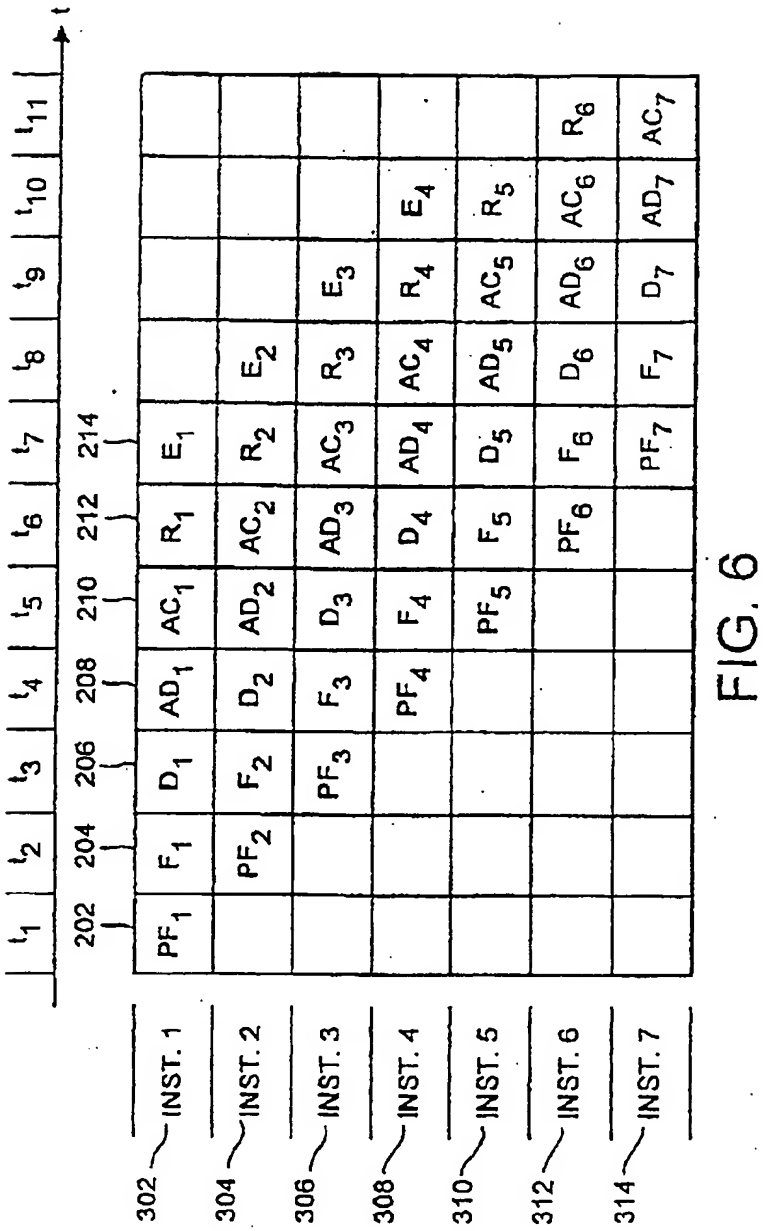

t
t1
t2
t3
t4
t5
t6
t7
t8
t9
t10
t11
202
204
206
208
210
212
214
INST. 1
INST. 2
INST. 3
INST. 4
INST. 5
INST. 6
INST. 7
302
304
306
308
310
312
314
PF1
F1
D1
AD1
AC1
R1
E1
PF2
F2
D2
AD2
AC2
R2
E2
PF3
F3
D3
AD3
AC3
R3
E3
PF4
F4
D4
AD4
AC4
R4
E4
PF5
F5
D5
AD5
AC5
R5
PF6
F6
D6
AD6
AC6
R6
PF7
F7
D7
AD7
AC7

FIG. 6

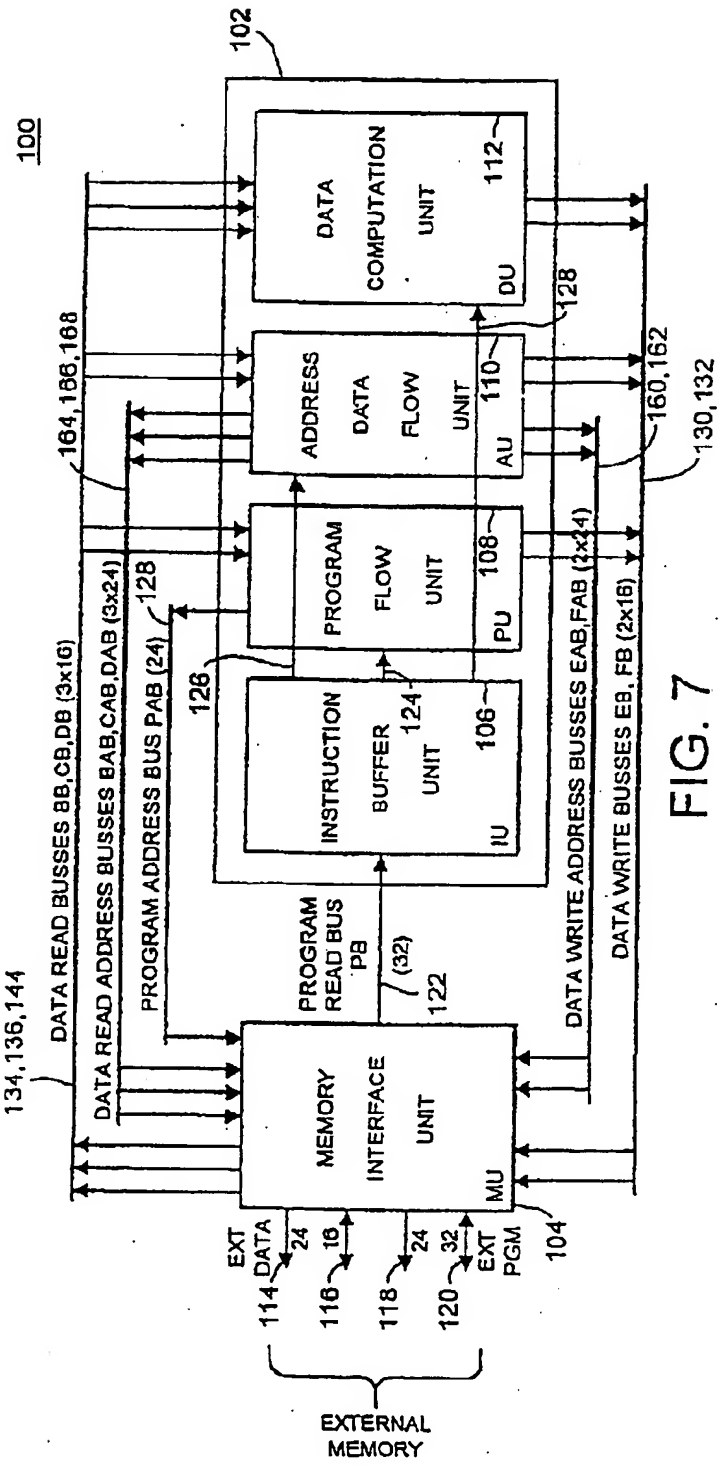

100
102
DATA COMPUTATION UNIT
112
DU
128
ADDRESS DATA FLOW UNIT
110
AU
160,162
130,132
164,166,168
PROGRAM FLOW UNIT
108
PU
INSTRUCTION BUFFER UNIT
124
106
IU
126
128
DATA READ BUSSES BB,CB,DB (3x16)
DATA READ ADDRESS BUSSES BAB,CAB,DAB (3x24)
PROGRAM ADDRESS BUS PAB (24)
DATA WRITE ADDRESS BUSSES EAB,FAB (2x24)
DATA WRITE BUSSES EB, FB (2x16)
134,136,144
PROGRAM READ BUS PB
(32)
122
MEMORY INTERFACE UNIT
MU
104
EXT DATA
24
16
24
32
EXT PGM
114
116
118
120
EXTERNAL MEMORY

FIG. 7

Instruction #1	Instruction #2	Example	Parallelism
Single Memory	Register	ACy = rnd (ACy + (Smem *ACx)) || dst = dst + sre	// enable
Single Memory	Control	ACy = rnd (ACy + (Smem *ACx)) || repeat (k8)	// enable
Single Memory	Stack	ACy = rnd (ACy + (Smem *ACx)) || push (sre)	// enable
Single Memory	Single Memory	ACx = rnd (Xmem *Xmem) || ACy = ACx + (Smem << DRx)	Soft Dual
Dual Memory	Register	ACx = (Xmem <<#16) + (Ymem <<#16) || DRy = DRx & k8	// enable
Dual Memory	Control	dbl (Ymem) - dbl (Xmem) || goto L8	// enable
Dual Memory	Stack	ACy = rnd (ACy + Smem *ACx)) || push (sre1 sre2)	// enable

FIG. 8

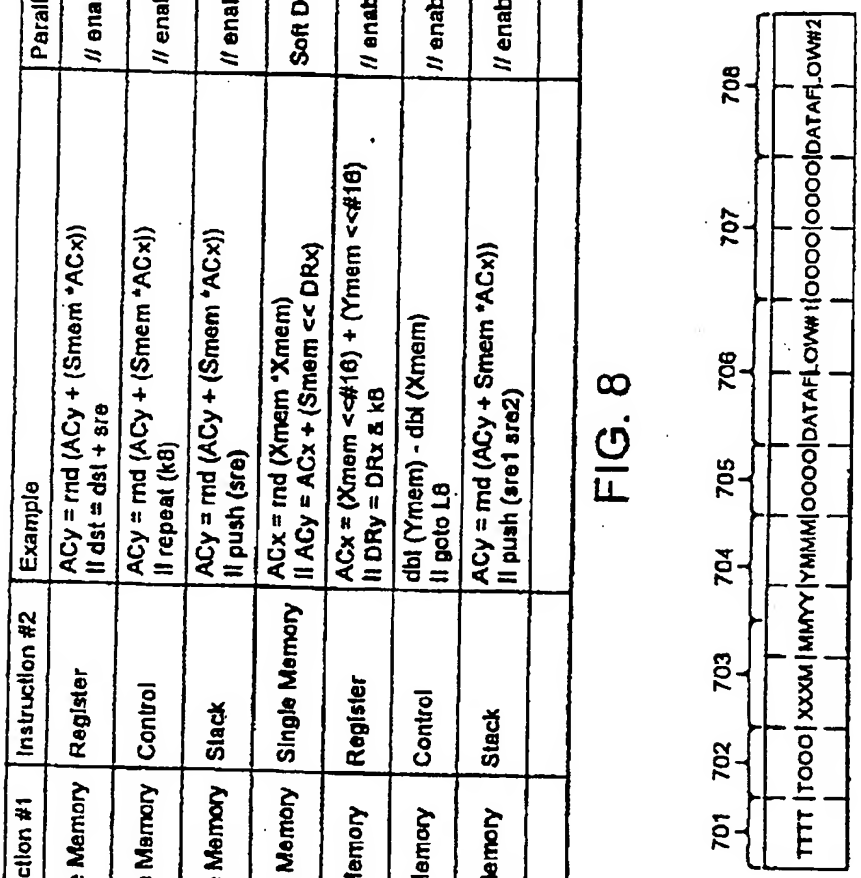


FIG. 10

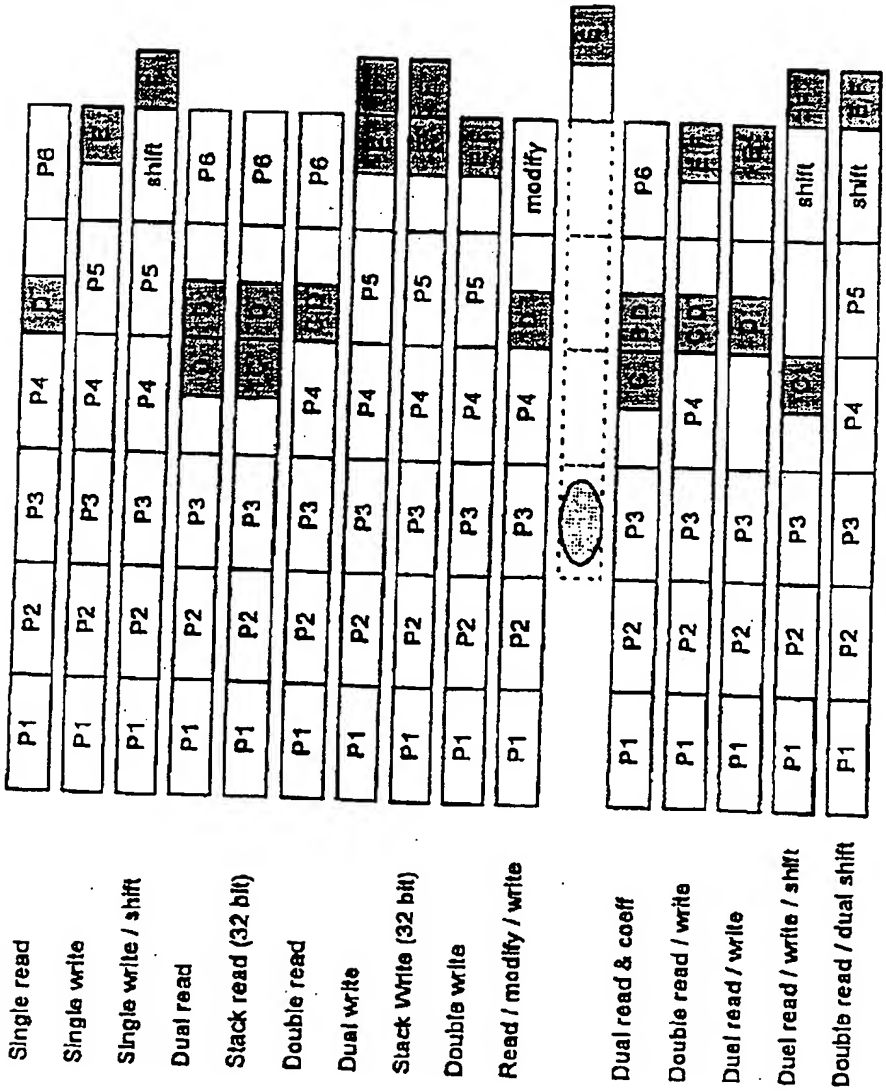

Single read
Single write
Single write / shift
Dual read
Stack read (32 bit)
Double read
Dual write
Stack Write (32 bit)
Double write
Read / modify / write
Dual read & coeff
Double read / write
Duel read / write
Duel read / write / shift
Double read / dual shift
P1
P2
P3
P4
P5
P6
shift
modify

FIG. 9

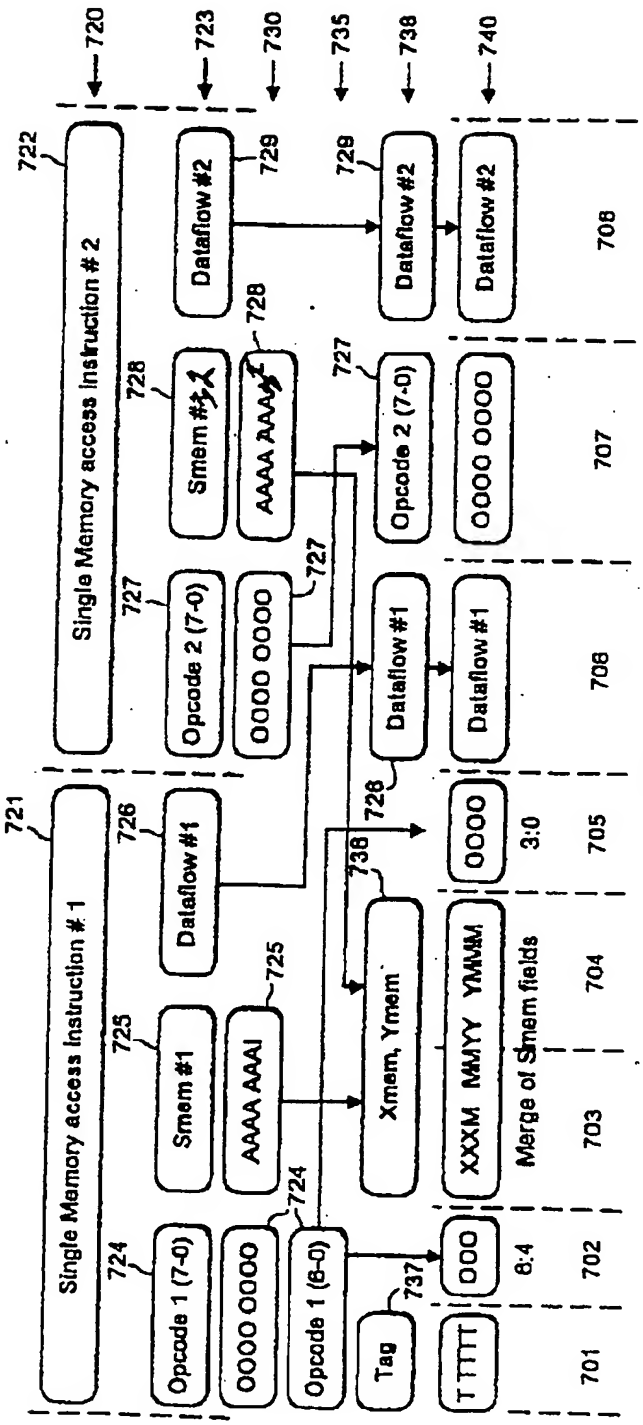

Single Memory access instruction # 1
Opcode 1 (7-0)
0000 0000
Opcode 1 (8-0)
Tag
TTTTT
000
8:4
701
702
Smem #1
AAAA AAAI
Xmem, Ymem
XXXM MMYY YMMM
Merge of Smem fields
0000
3:0
703
704
705
Dataflow #1
Opcode 2 (7-0)
0000 0000
Dataflow #1
Dataflow #1
708
Smem #2
AAAA AAAA
Opcode 2 (7-0)
0000 0000
Dataflow #2
Dataflow #2
Dataflow #2
707
708
Single Memory access instruction # 2
720
721
722
723
726
727
728
729
730
735
738
740

FIG. 11

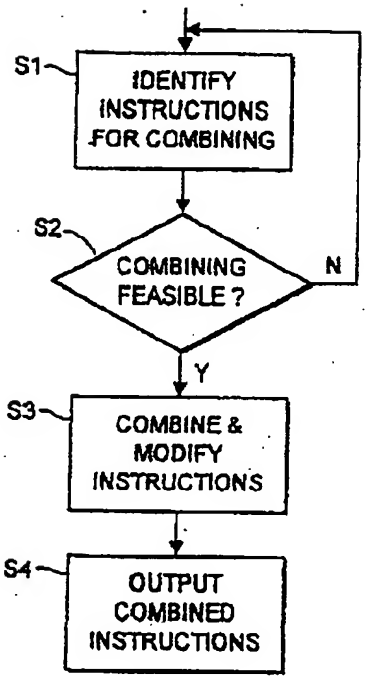

S1
IDENTIFY INSTRUCTIONS FOR COMBINING
S2
COMBINING FEASIBLE ?
N
Y
S3
COMBINE & MODIFY INSTRUCTIONS
S4
OUTPUT COMBINED INSTRUCTIONS

FIG. 12

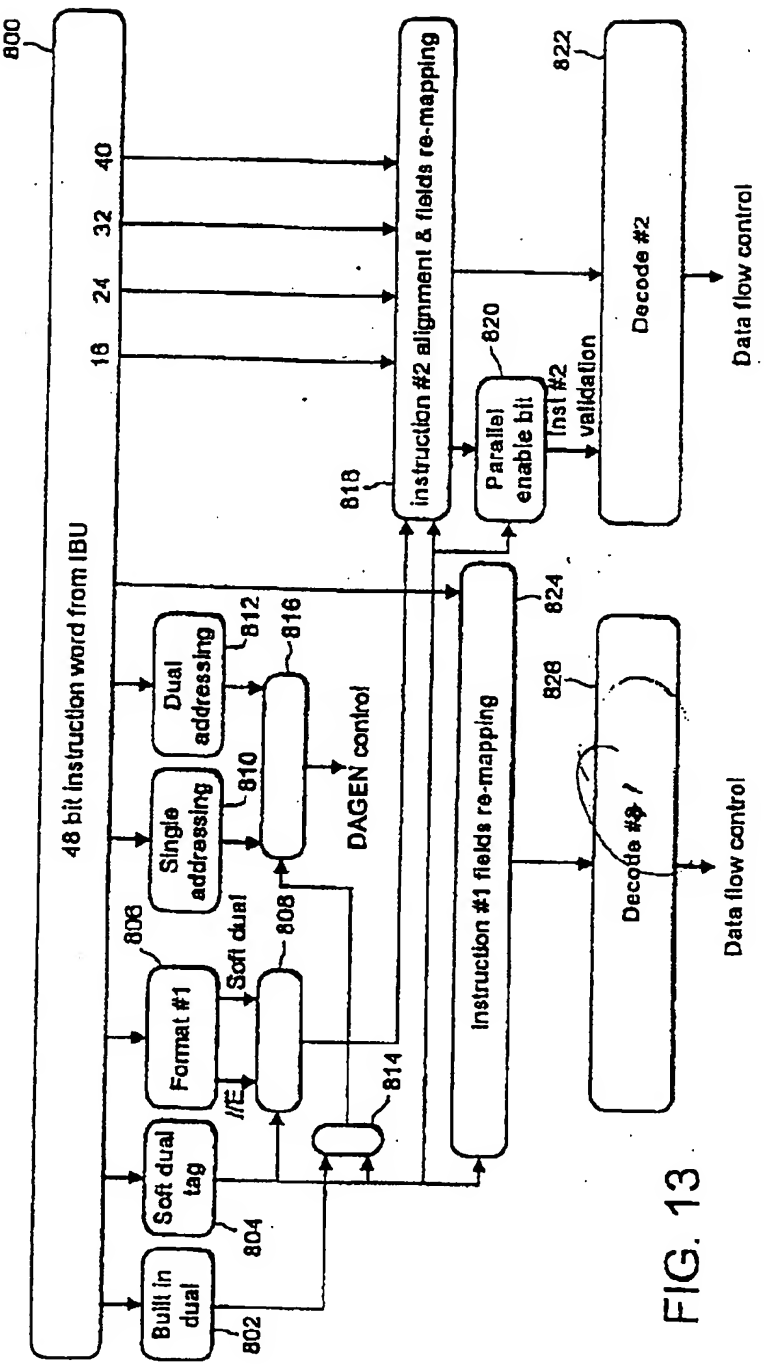

800
48 bit instruction word from IBU
40
32
24
18
Built in dual
802
Soft dual tag
804
Format #1
806
Soft dual
//E
808
Single addressing
810
Dual addressing
812
814
816
DAGEN control
818
instruction #2 alignment & fields re-mapping
Parallel enable bit
820
Inst #2 validation
822
Decode #2
Data flow control
824
Instruction #1 fields re-mapping
826
Decode #1
Data flow control

FIG. 13

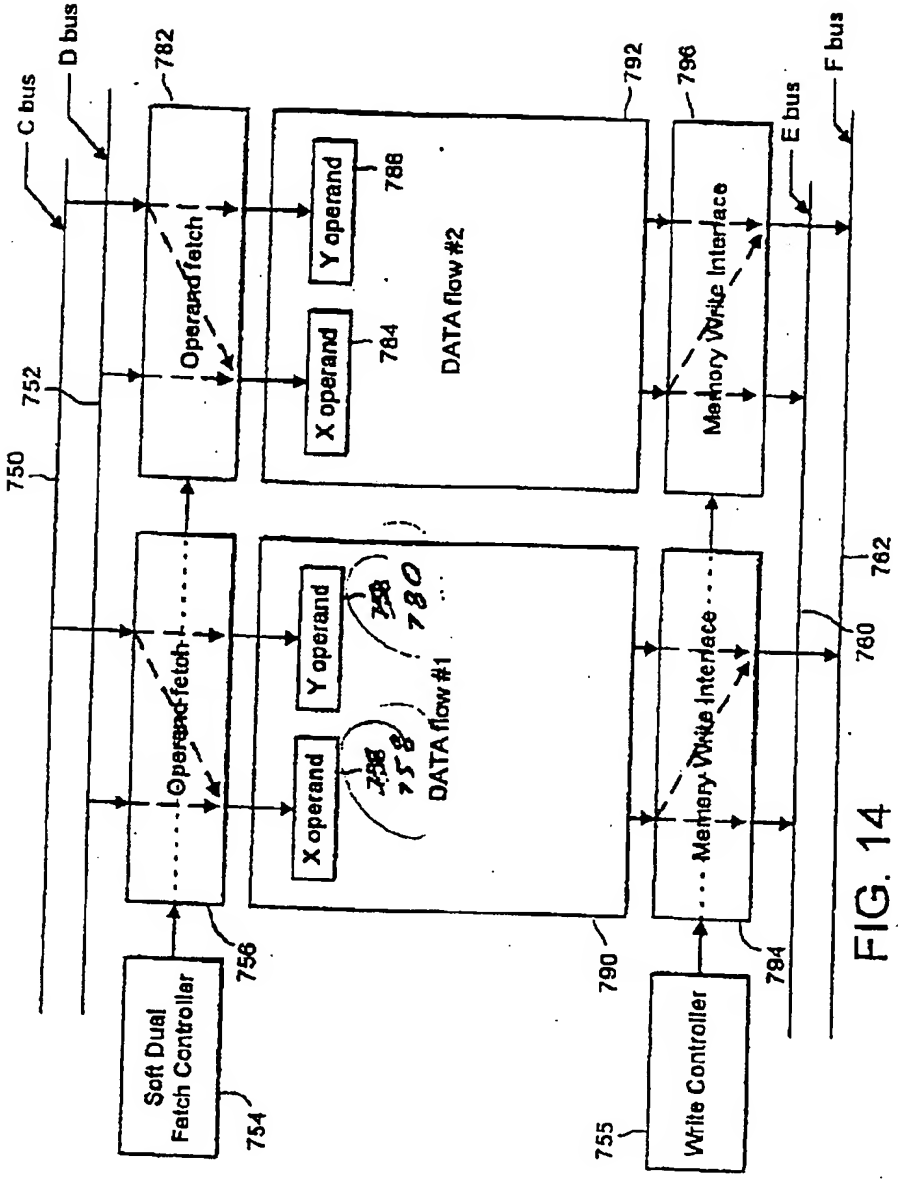

C bus
D bus
782
792
796
F bus
E bus
Y operand
788
Operand fetch
DATA flow #2
X operand
784
Memory Write Interface
752
750
762
780
Y operand
Operand fetch
DATA flow #1
X operand
Memory Write Interface
790
784
756
Soft Dual Fetch Controller
754
755
Write Controller

FIG. 14

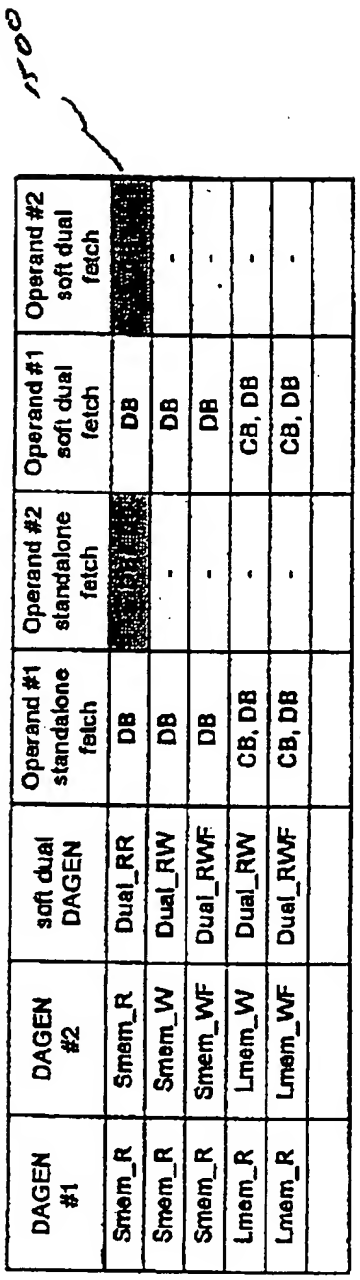

1500
DAGEN #1
DAGEN #2
soft dual DAGEN
Operand #1 standalone fetch
Operand #2 standalone fetch
Operand #1 soft dual fetch
Operand #2 soft dual fetch
Smem_R Smem_R Dual_RR DB - DB -
Smem_R Smem_W Dual_RW DB - DB -
Smem_R Smem_WF Dual_RWF DB - DB -
Lmem_R Lmem_W Dual_RW CB, DB - CB, DB -
Lmem_R Lmem_WF Dual_RWF CB, DB - CB, DB -

FIG. 15

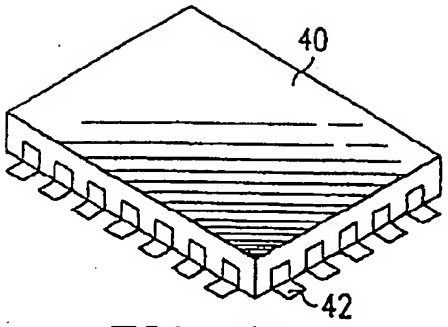


FIG. 16.

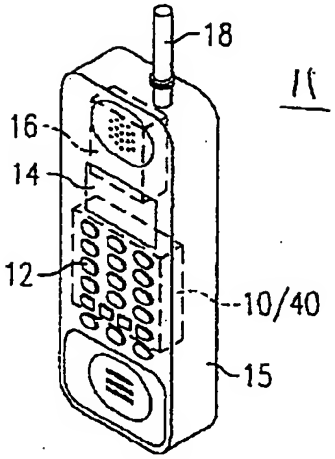


FIG. 17

1. Abstract

A processing engine 10 includes an instruction buffer 502 operable to buffer single and compound instructions pending execution. A decode mechanism is configured to decode instructions from the instruction buffer. The decode mechanism is arranged to respond to a predetermined tag in a tag field of an instruction, which predetermined tag is representative of the instruction being a compound instruction formed from separate programmed memory instructions. The decode mechanism is operable in response to the predetermined tag 726 to decode at least first data flow control for a first programmed instruction 721 and second data flow control 729 for a second programmed instruction 722. The use of compound instructions enables effective use of the bandwidth available within the processing engine. A soft dual memory instruction can be compiled from separate first and second programmed memory instructions. A compound address field 738 of the predetermined compound instruction can be arranged at the same bit positions as the address field for a hard compound memory instruction, that is a compound instruction which is programmed. In this case the decoding of the addresses can be started before the operation code of the instructions have been decoded. To reduce the number of bits in the compound instruction, addressing can be restricted to indirect addressing and the operation codes for at least the first instruction can be reduced in size. In this way, the compound instruction can be arranged to have the same number of bits in total as the sum of the bits of the separate programmed instructions.

2 Representative Drawing

Fig. 11